Trabajo de Final de Grado

Grado en Ingeniería en Tecnologías Industriales

Diseño y desarrollo del sistema de control de un vehículo de tres ruedas

MEMORIA

Autor: Mikel, Aceldegui Bernal
Director: Rosa, Rodríguez Montañés

Convocatoria: Junio 2019

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Barcelona, a 20 de junio de 2019. **(FIRMA)**

ETSEIB

Dedicatoria

Este trabajo de final de grado está dedicado a:

Mis padres y abuelos:

Quienes me apoyaron moral, emocional y económicamente a lo largo de toda mi vida estudiantil y que me ofrecieron todo lo necesario para terminar mis estudios.

Mis amigos:

Quienes fueron un apoyo social y emocional durante mi vida como estudiante.

Mis profesores:

Que compartieron su tiempo y conocimiento para formarme profesionalmente.

Resumen

Este proyecto consiste en el diseño y desarrollo del sistema de control de un vehículo de tres ruedas. El sistema electrónico implementado está basado principalmente un microcontrolador PIC, dos motores de corriente continua y diferentes tipos de periféricos. Para la comunicación entre el usuario y el sistema electrónico se ha diseñado una aplicación para sistema operativo *Android* basada en el intercambio de señales en protocolo *Bluetooth*.

El vehículo contará con dos modos diferentes de conducción, el control mediante giroscopio y el control manual. En el modo de conducción manual el usuario tendrá a su disposición una serie de botones capaces de controlar plenamente la trayectoria y velocidad del vehículo. En el modo de conducción por giroscopio, el control del vehículo se realiza a través de la inclinación del dispositivo *smartphone* que esté siendo utilizado.

La comunicación del usuario con el vehículo se realizará vía *Bluetooth* gracias a un módulo hardware que incorporará el coche y a un teléfono móvil con sistema operativo Android. Previamente, al dispositivo *smartphone* se le instalará una aplicación móvil que también se desarrollará en el proyecto.

Para este trabajo de final de grado se utilizará un microcontrolador de 8 bits de la familia PIC del fabricante Microchip. Además, los softwares utilizados en la programación serán MPLAB IDE para el microcontrolador (lenguaje C) y AppInventor2 del MIT para el desarrollo de la aplicación para dispositivos *smartphones* Android.

En este documento se realizará un análisis detallado de los componentes utilizados para la construcción del vehículo, así como los pasos previos de análisis y compresión del funcionamiento de éstos. También incluirá la explicación del diseño y desarrollo de la aplicación móvil para controlar el vehículo y del programa implementado en el microcontrolador.

Sumario

ANEXOS

Anexo 1. CÓDIGO FUENTE C *Programa Principal*

Anexo 2. CÓDIGO FUENTE C *Archivo de Cabecera "uart.h"*

Anexo 3. PROGRAMACIÓN POR BLOQUES *Aplicación Móvil de AppInventor 2*

Anexo 4. DATASHEET *PIC16(L)F722A/723A*

Anexo 5. DATASHEET *PICkit 3*

Anexo 6. DATASHEET *Módulo Bluetooth HC-05*

Anexo 7. DATASHEET *L293x Motor Shield*

Anexo 8. DATASHEET *Módulo de Ultrasonidos HC-SR04*

1. Glosario

Microcontrolador: Circuito integrado en un solo chip que contiene una unidad central de procesamiento (CPU), unidades de memoria (RAM y ROM), puertos de entrada/salida y periféricos.

Microprocesador o procesador (CPU): Circuito integrado central de un sistema informático. Es el encargado de realizar las operaciones de cálculo y control del microcontrolador, recibiendo información y dando órdenes a los demás elementos.

RISC (del inglés ***R**educed **I**nstruction **S**et **C**omputer***):** Arquitectura usada generalmente en microcontroladores o microprocesadores que se caracteriza por tener un conjunto de instrucciones simples y generales, lo que le permite tener menos ciclos de reloj por instrucción. Además, en esta arquitectura solo las instrucciones de carga y almacenamiento acceden a la memoria de datos.

Arquitectura Harvard: La Unidad Central de Proceso (CPU) está conectada a dos memorias, memoria de programa (ROM) y memoria de datos (RAM). Ambos buses son totalmente independientes y pueden tener distinta longitud, esto permite acceder a la memoria de datos para completar la ejecución de una instrucción y al mismo tiempo leer la siguiente instrucción a ejecutar.

ROM (del inglés ***R**ead **O**nly **M**emory*): Circuito integrado de memoria de solo lectura que permite almacenar instrucciones y datos de forma permanente.

RAM (del inglés ***R**andom **A**ccess **M**emory*): Memoria de acceso aleatorio en la cual se puede leer y escribir en una posición de memoria con un tiempo de espera igual para cada posición.

EEPROM (del inglés ***E**lectrically **E**rasable **P**rogrammable **R**ead **O**nly **M**emory*)**:** Es un tipo de memoria ROM que puede ser programada y reprogramada eléctricamente.

FLASH: Memoria derivada de la memoria EEPROM orientada al almacenamiento de grandes cantidades de datos permitiendo la lectura y escritura de múltiples posiciones de memoria, lo que otorga una mayor velocidad de funcionamiento.

PWM (del inglés ***P**ulse **W**idth **M**odulation*)**:** Técnica en la que se modifica el ciclo de trabajo de una señal periódica, ya sea para transmitir información a través de un canal o para controlar la cantidad de energía que se envía a una carga.

Giroscopio de tipo MEMS (Sistemas **M**icro **E**lectro **M**ecánicos)**:** Giroscopios que incluyen los *smartphones*, son capaces de detectar cambios en la posición del teléfono con mucha exactitud y convierten el movimiento en una señal eléctrica de baja corriente que es amplificada e interpretado por un microcontrolador.

USART (del inglés ***U****niversal* ***S****ynchronous* ***A****synchronous* ***R****eceiver* ***T****ransmitter***):** Es un dispositivo que realiza la transmisión o recepción de datos secuenciales entre dos nodos de comunicación de forma asíncrona o síncrona.

UART (del inglés *Universal Asynchronous Receiver Transmitter**):*** Es la comunicación mediante el módulo USART de manera asíncrona.

SPRITE: Máscara o Imagen de un objeto que tiene la capacidad de colisión y puede ser gestionado de manera independiente al canvas que lo contiene.

CANVAS: Es un panel rectangular bidimensional sensible al tacto en que se pueden realizar dibujos y puede mover los sprites.

SLIDER-BAR: Un *slider-bar* es un elemento de las interfaces gráficas que permite seleccionar al usuario un valor moviendo un indicador con forma de barra.

2. Prefacio

2.1. Origen del proyecto

El autor de este proyecto se encuentra cursando su último semestre del grado en ingeniería en tecnologías industriales en la Universidad Politécnica de Cataluña, mediante el programa de movilidad nacional SICUE.

En su universidad de origen (Universidad Pública de Navarra, UPNA) tras haber completado con éxito la mayoría de las asignaturas, los ámbitos de la electrónica y el control de sistemas son los que más desataron su atención, especializándose así en el tercer y cuarto curso por la rama de optativas de electrónica. Es por ello que el proyecto que desarrolla combina tanto la electrónica como el control, realizando el hardware del sistema electrónico necesario para el vehículo de tres ruedas y su manejo mediante el teléfono móvil del usuario.

2.2. Motivación

Durante este último año el autor de este trabajo estuvo trabajando para su universidad origen en el diseño y desarrollo de un espectrofotómetro de bajo coste mediante Raspberry PI 3, por lo que tuvo que abarcar tanto la parte de diseño del sistema electrónico como la parte de programación y procesado de señal. Gracias a ello, ha tenido la oportunidad de ver desde dentro la innovación que supone la electrónica para el futuro y las oportunidades de trabajo que aporta.

En base a lo anterior, la primera intención al decidir el ámbito que abarcaría el proyecto de final de grado fue el electrónico combinado con el control, como podía ser la construcción de un péndulo invertido, pero tras solicitar la opinión a la tutora, vista la complejidad y el coste que supondría hacer este tipo de sistemas, se optó por construir algo más asequible económicamente y adaptado al nivel de conocimientos adquirido durante los estudios de grado.

Finalmente, otra gran fuente de motivación para el autor ha sido el poder aplicar en un desarrollo funcional los conceptos teóricos recibidos previamente durante su formación universitaria, además de poder ampliar sus conocimientos técnicos. Así mismo, el proyecto realizado constituye una plataforma que será utilizada en el futuro de una de las asignaturas que imparte el departamento de ingeniería electrónica en la ETSEIB y está relacionada con proyectos que controlan el movimiento de vehículos de tres ruedas, mediante comunicación con *Smartphones*.

2.3. Requerimientos previos

Para llevar a cabo la realización del proyecto eran necesarios una serie de conocimientos previos tanto de electrónica y análisis de circuitos como de programación dado que en el proyecto se combinan estos dos ámbitos.

El autor de este proyecto en su universidad origen ya había realizado algunas asignaturas relacionadas con programación en *Python* y en *Pascal*¸ por lo tanto el aprendizaje del lenguaje C no ha sido tan complicado durante el desarrollo del proyecto.

Los conocimientos de electrónica y análisis de circuitos se adquirieron más en profundidad durante su tercer año de grado con la especialidad de electrónica, donde también pudo conocer la familia de microcontroladores PIC y cómo funcionaban a grandes rasgos. Respecto a los demás componentes del vehículo, el autor ya había tenido una toma de contacto con diferentes sensores y actuadores parecidos o similares durante el desarrollo de su grado, lo que le ha servido de base sólida para analizar y utilizar los nuevos dispositivos que ha necesitado.

Respecto a la parte de programación de la aplicación para *Smartphone*, el aprendizaje ha sido autónomo, tras un análisis previo del software más adecuado para desarrollarlo y, de este modo, realizar una comunicación móvil/vehículo óptimo.

3. Introducción

En este apartado se planteará el enfoque del proyecto que se ha desarrollado, dando a conocer tanto los objetivos que quieren lograrse como el alcance del proyecto. De esta forma se tendrá una mejor visión de los ámbitos y las diferentes partes a analizar, diseñar e implementar.

3.1. Objetivos del proyecto

3.1.1. Objetivo General

El objetivo principal de este proyecto es diseñar y desarrollar un sistema electrónico basado en microcontrolador, capaz de interactuar con unos estímulos externos recibidos, y realizar las acciones correspondientes sobre los dos motores DC disponibles, con la finalidad de definir la trayectoria y velocidad de un vehículo de tres ruedas.

3.1.2. Objetivos Específicos

El objetivo general da lugar a una serie de objetivos específicos que se listan a continuación:

- Aplicación de los conocimientos de electrónica y programación adquiridos durante el grado en un caso real de programación de microcontroladores.
- Montaje del prototipo del vehículo de tres ruedas y conexionado de los componentes electrónicos (sensores/actuadores) elegidos mediante un análisis previo.
- Desarrollo de una aplicación móvil para *smartphone* con una interfaz gráfica sencilla para la interacción del usuario con el movimiento del vehículo.
- Elaboración de la programación de varios modos de conducción con los que pueda interaccionar el usuario desde la aplicación móvil y de esta manera dar una mayor flexibilidad al control del vehículo.

3.2. Alcance del proyecto

Este proyecto abarca desde la selección y montaje de los diferentes módulos del proyecto, hasta su programación y puesta en funcionamiento para el control de vehículo, incluyendo también el ensamblaje previo del chasis que lo forma.

Por otro lado, en este apartado también se detallarán los diferentes condicionantes y limitaciones que se imponen al trabajo.

- Material Utilizado: Dado que el presupuesto del proyecto no era elevado, el material utilizado es el que se disponía en el laboratorio de electrónica de la ETSEIB, al cual se le dio acceso al estudiante.

- Tipo de Comunicación entre móvil/vehículo: La comunicación entre el dispositivo *smartphone* y el microcontrolador se realiza mediante comunicación vía *Bluetooth*, por ser un protocolo de comunicación extendido y al alcance de la mayoría de los usuarios

- Aplicación Móvil: La aplicación móvil se diseña para un sistema operativo *Android* dado que actualmente es el más utilizado en los dispositivos *smartphones* y permite tener acceso a la creación de aplicaciones gratuitas.

- Lenguaje de programación del microcontrolador: Para la realización del código fuente del programa que contendrá el PIC se utiliza lenguaje C, dado que es el que se utiliza en la programación de este tipo de microcontroladores, además de ser sencillo de utilizar y aprender.

- Software para el desarrollo de la Aplicación Móvil: Para la programación de la aplicación móvil se utiliza el software *MIT App Inventor 2*. Este software permite una programación intuitiva y sencilla mediante el uso bloques, además de dar la posibilidad de realizar interfaces gráficas de forma sencilla.

- Conocimientos de electrónica y programación del autor: El autor consta de una formación previa sobre estos ámbitos. Estos conocimientos son limitados, es decir, tanto los módulos electrónicos utilizados como el tipo del vehículo a controlar son sencillos y su uso está orientado al aprendizaje.

4. Especificaciones del sistema

En este apartado se explicará en detalle las diferentes características y especificaciones presentes en el vehículo desarrollado, comenzando por los tipos de conducción y continuando por el microcontrolador utilizado y los motores de corriente continua que permitirán su desplazamiento.

El vehículo cuenta con un **chasis** principal y con **tres ruedas.** Dos de las ruedas irán conectadas a motores de corriente continua para dar movimiento al vehículo y la tercera rueda servirá como soporte para equilibrar la estructura del chasis situándose en la parte trasera de éste, por lo que es una rueda libre que depende de las otras dos. En la Figura 1 pueden visualizarse las partes del vehículo comentadas.

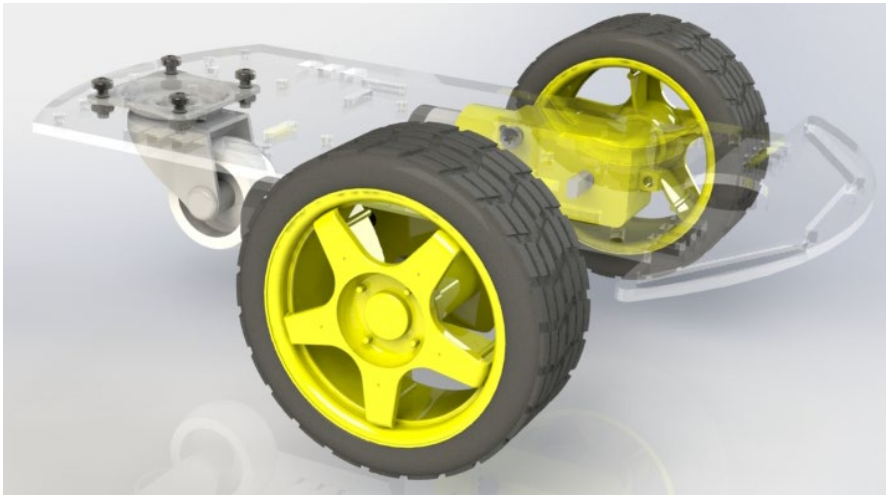


Figura 1: Previsualización 3D del vehículo del proyecto mediante SolidWorks. Fuente: Propia

4.1. Movimientos y modos de conducción

El vehículo ha de poder seguir cualquier trayectoria que el usuario le indique, sobre una superficie plana y, para ello, la aplicación Android diseñada como parte del proyecto contará con dos modos de conducción diferentes, el modo denominado Giroscopio y el modo denominado Libre.

Ambos modos requieren de un teléfono *smartphone* Android con la aplicación del proyecto instalada, debido a que con esta aplicación el usuario podrá establecer la conexión *Bluetooth* entre móvil/vehículo y seleccionar el modo de conducción.

4.1.1. Modo Giroscopio

En el modo Giroscopio la interacción se realizará mediante el giroscopio de tipo MEMS incorporado en el teléfono *smartphone* y, en función de su inclinación, determinará el tipo de movimiento que se quiera realizar. En la programación implementada sobre el microcontrolador, se cuenta con un ciclo de trabajo de señal PWM individual para cada motor de corriente continua, por lo que se podrá controlar la velocidad y el sentido de forma independiente en ambos motores.

De los tres ángulos de giro de los cuales informa el giroscopio (Roll, giro en X – Pitch, giro en Y – Yaw, giro en Z, Figura 2), en este modo de conducción se utilizarán solamente dos, Roll y Pitch, por resultar el tercero, innecesario.

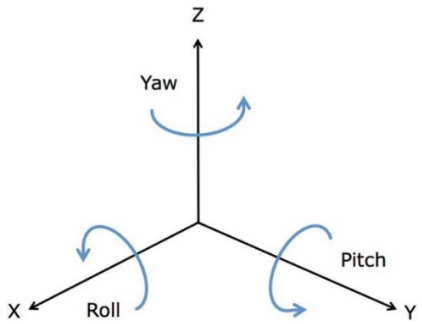


Figura 2: Ángulos obtenidos del giroscopio tipo MEMS del teléfono smartphone. Fuente: https://www.researchgate.net/figure/Average-roll-pitch-and-yaw-angles_fig2_262055313

4.1.2. Modo Libre

En el modo Libre, el usuario tendrá disponible los cinco botones con las opciones básicas de trayectoria: hacia adelante, hacia atrás, giro a derechas, giro a izquierdas y paro. Además, contará con una *slider-bar* para modificar el ciclo de trabajo de ambos motores a la vez y tener un control directo en la velocidad del vehículo.

Finalmente, este modo también incluirá un sensor de ultrasonidos capaz de detectar distancias entre el vehículo y los objetos del entorno, de tal forma que el coche se detenga antes de que colisiones con estos.

4.2. Microcontrolador

El microcontrolador usado en el proyecto es un PIC16F723A de 8bits y fabricante Microchip. Se ha utilizado este microcontrolador dado que cuenta con características adecuadas a las necesidades del proyecto y, concretamente, cuenta con dos módulos CCP para controlar los motores del vehículo, un módulo UART para la comunicación vía *Bluetooth* con el teléfono y una frecuencia de oscilación interna seleccionable de hasta 16 MHz.

Es el componente más importante del vehículo dado que es el encargado de interpretar las órdenes externas del usuario y procesarlas mediante el programa guardado en él, para finalmente generar las correspondientes acciones de salida y gobernar así el movimiento del coche.

4.3. Motores DC

Los dos motores de corriente continua que incorpora el vehículo son los actuadores más importantes, dado que se encargan de generar el movimiento solicitado por el usuario. Ambos motores son independientes entre sí y requieren de una intensidad de corriente con picos de corriente de hasta 200 mA. El microcontrolador utilizado cuenta con una intensidad de salida máxima por los pines I/O de 25 mA (), por lo tanto, se requiere el uso de un **amplificador de corriente** para su correcto funcionamiento, tal y como se detallará más adelante.

Maximum output current sunk by any I/O pin	25 mA
Maximum output current sourced by any I/O pin	25 mA

Figura 3: Máxima corriente de salida suministrada por los pines I/O del PIC16F723A. Fuente: [1]

4.4. Voltaje de Alimentación

Todos los elementos electrónicos incluidos en el diseño del proyecto pueden alimentarse a un nivel de tensión continua de valor standard de 5 voltios y, por lo tanto, éste ha sido el nivel de tensión elegido para la implementación del proyecto

5. Arquitectura del sistema (Esquema Funcional)

En este apartado se mostrarán y explicarán los diferentes esquemas de conexionado entre los módulos para lograr el funcionamiento deseado del proyecto.

El esquema principal del proyecto se puede dividir internamente en dos partes. La primera de ellas corresponde al esquema funcional relativo a la implementación del modo de conducción Giroscopio, y la segunda parte corresponde al esquema funcional del modo Libre. El modo de conducción en el que se encuentra funcionando el vehículo depende únicamente de la selección realizada por el usuario a través de la aplicación del dispositivo *smartphone*.

5.1. Esquema funcional del modo giroscopio

El conexionado de los módulos utilizados en el modo de conducción del Giroscopio se muestra en la Figura 4.

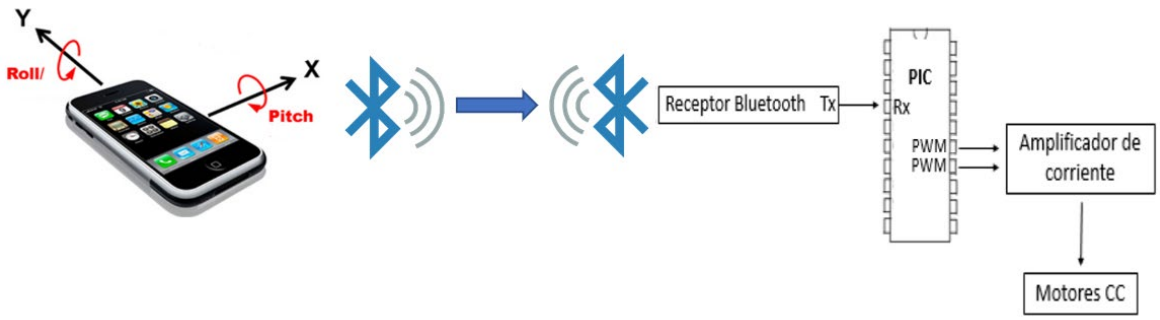


Figura 4: Esquema funcional de la conexión de los módulos en el modo giroscopio. Fuente: Propia

Los datos relativos a los ángulos captados por el giroscopio del teléfono móvil son enviados vía *Bluetooth* desde el teléfono (“Maestro”) al módulo receptor externo (“Esclavo”) situado en el vehículo. La función de este módulo es transmitir estos datos en modo serie al PIC quien los interpreta gracias al programa que implementa y actúa en consecuencia.

5.2. Esquema funcional del modo libre

De igual forma que en el apartado anterior, las conexiones utilizadas en el modo de conducción Libre se muestran en la Figura 5.

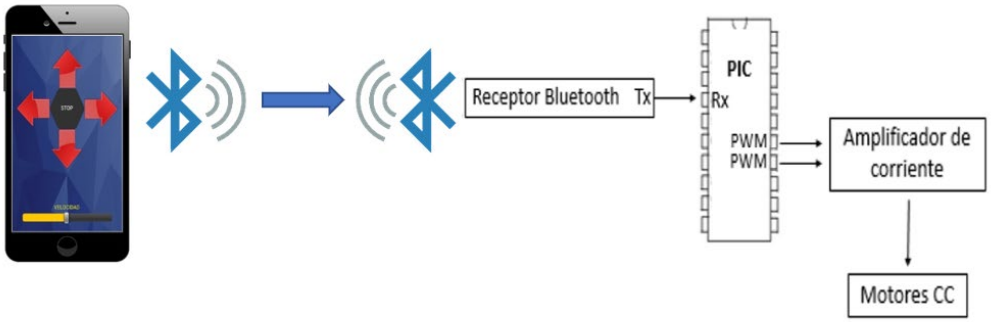


Figura 5: Esquema funcional de la conexión de los módulos en el modo giroscopio. Fuente: Propia

El usuario tendrá a su disposición una serie de botones y una *slider bar*. La funcionalidad que use mediante esta serie de opciones determinará la trama de datos enviada vía *Bluetooth* al receptor del vehículo. Una vez es recibida e interpretada esta información por el PIC, gracias al programa cargado en él, se enviarán las acciones correspondientes a los motores de corriente continua.

Los módulos utilizados en el modo de conducción Libre son los mismos que en el modo Giroscopio incluyendo además el sensor de ultrasonidos para la detección de objetos. Las diferencias entre ambos modos (Libre y Giroscopio) son la programación para actuar sobre los motores, los datos de referencia usados, la capacidad de detener el vehículo antes un obstáculo en el modo Libre y la interfaz de usuario utilizada en la aplicación móvil.

6. Microcontrolador PIC16F723A

Es un microcontrolador de 8 bits del fabricante *Microchip* y destaca por tener un tipo de tecnología con consumos extremadamente pequeños (*"Extreme Low Power MCUs"*) idónea para alargar la vida de la batería del vehículo considerado en este proyecto. Este modelo de microcontrolador es de gama media con tecnología CMOS y cuenta con 28 pines configurables [1].

En el interior del microcontrolador se encuentran la CPU (del inglés *Central Procesing Unit*), unidades de memoria, periféricos de entrada/salida. Todos ellos conectados entre sí mediante distintos tipos de buses.

Figura 6: Microcontrolador PIC 16F723A. Fuente: https://sa.rsdelivers.com/

Una breve descripción de los tipos de módulos y memorias que implementa el PICF723A se muestra en la Figura 7.

Device	Data Sheet Index	Program Memory Flash (words)	Data SRAM (bytes)	High-Endurance Flash Memory (bytes)	I/O's(2)	8-bit ADC (ch)	CapSense (ch)	Timers (8/16-bit)	AUSART	SSP (I^2C/SPI)	CCP	Debug(1)	XLP
PIC16(L)F723A	(3)	4096	192	0	25	11	8	2/1	1	1	2	I	Y

Figura 7: Características del PIC16F723A. Fuente:[1]

6.1. Diagrama de bloques del micro

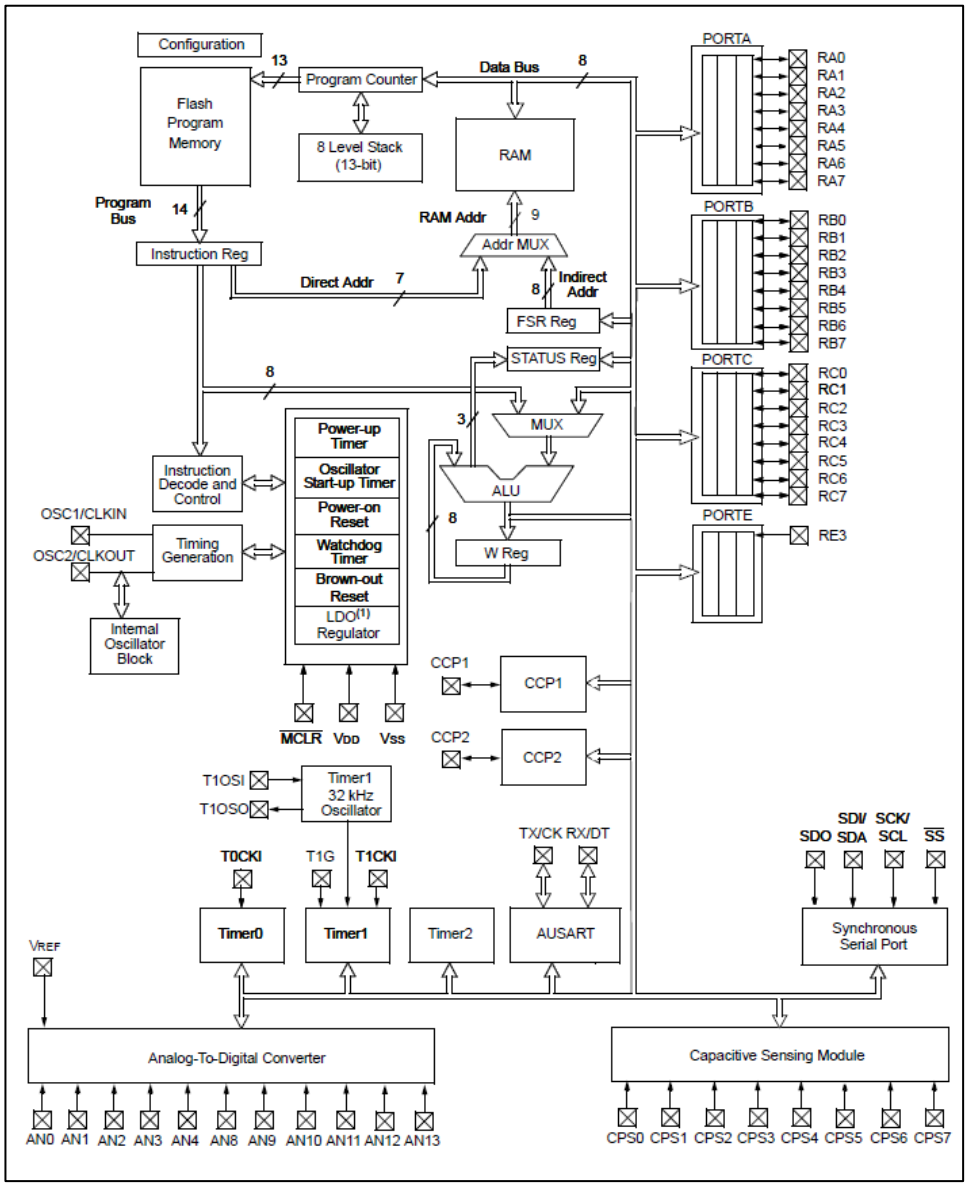


Figura 8: Diagrama de bloques del PIC16F723A. Fuente: [1]

En el diagrama de bloques de la Figura 8, se puede observar varios aspectos importantes de su arquitectura que se listan a continuación:

- Bus de direcciones de datos de 9 bits (*SRAM de 512 x 8*)
- Bus de datos de 8 bits
- Bus de direcciones de programa con 13 bits (*FLASH de 8K x 14)*
- Bus de instrucciones de programa de 14 bits
- 8 niveles de almacenamiento en la Pila

- 4 puertos I/O: A (8 pines), B (8 pines), C (8 pines), E (1 pin)
- Unidad aritmético lógica ALU de 8 bits
- Tipos de direccionamiento Directo e Indirecto
- Módulo AUSART
- Timers 0, 1 y 2
- 2 Puertos CCP (Captura/Comparación/PWM)

6.2. Memoria de programa, FLASH

Este tipo de memorias son de sólo lectura, dado que almacenan los códigos binarios correspondientes a las instrucciones del programa que se ejecuta en el microcontrolador.

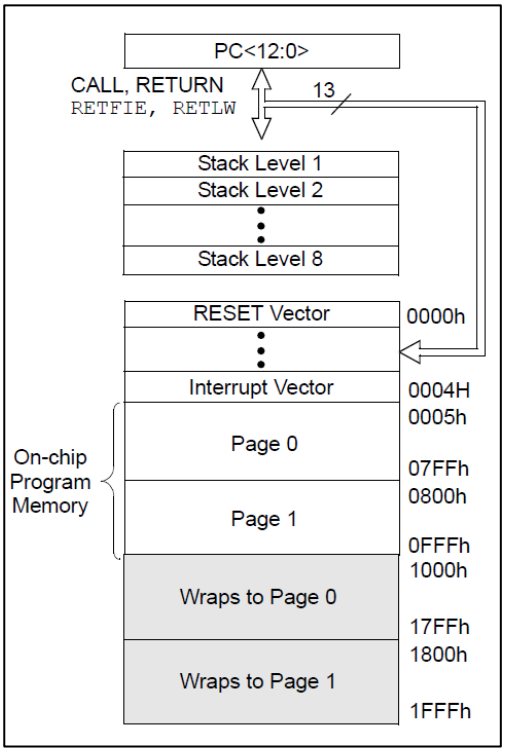


Figura 9: Memoria de programa FLASH del PIC16F723A. Fuente:[1]

En la Figura 9 puede observarse que el PIC16F723A cuenta con una memoria *FLASH* con un espacio direccionable de 8K (2^{13} bits) de los cuales solamente 4K están implementados. Para acceder a las diferentes páginas y posiciones de la memoria de programa se utiliza el PC (Program Counter), el cual es un registro interno del microcontrolador que indica la posición de la siguiente instrucción a ejecutar.

Por último, también se observa que el *vector RESET* y el *vector de atención a las interrupciones* se implementan en las posiciones 0000h y 0004h, respectivamente.

6.3. Memoria de datos, SRAM

La memoria de datos es de tipo SRAM (*"Static RAM"*), de modo que es una memoria volátil de lectura y escritura que almacena todos los datos que se manejan en el programa.

Bank 0		Bank 1		Bank 2		Bank 3	
Indirect addr.(*)	00h	Indirect addr.(*)	80h	Indirect addr.(*)	100h	Indirect addr.(*)	180h
TMR0	01h	OPTION	81h	TMR0	101h	OPTION	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h		105h	ANSELA	185h
PORTB	06h	TRISB	86h		106h	ANSELB	186h
PORTC	07h	TRISC	87h		107h		187h
	08h		88h	CPSCON0	108h		188h
PORTE	09h	TRISE	89h	CPSCON1	109h		189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch	PMDATL	10Ch	PMCON1	18Ch
PIR2	0Dh	PIE2	8Dh	PMADRL	10Dh	Reserved	18Dh
TMR1L	0Eh	PCON	8Eh	PMDATH	10Eh	Reserved	18Eh
TMR1H	0Fh	T1GCON	8Fh	PMADRH	10Fh	Reserved	18Fh
T1CON	10h	OSCCON	90h		110h		190h
TMR2	11h	OSCTUNE	91h		111h		191h
T2CON	12h	PR2	92h		112h		192h
SSPBUF	13h	SSPADD/SSPMSK	93h		113h		193h
SSPCON	14h	SSPSTAT	94h		114h		194h
CCPR1L	15h	WPUB	95h		115h		195h
CCPR1H	16h	IOCB	96h		116h		196h
CCP1CON	17h		97h		117h		197h
RCSTA	18h	TXSTA	98h		118h		198h
TXREG	19h	SPBRG	99h		119h		199h
RCREG	1Ah		9Ah		11Ah		19Ah
CCPR2L	1Bh		9Bh		11Bh		19Bh
CCPR2H	1Ch	APFCON	9Ch		11Ch		19Ch
CCP2CON	1Dh	FVRCON	9Dh		11Dh		19Dh
ADRES	1Eh		9Eh		11Eh		19Eh
ADCON0	1Fh	ADCON1	9Fh		11Fh		19Fh
General Purpose Register 96 Bytes	20h – 7Fh	General Purpose Register 80 Bytes	A0h – EFh	General Purpose Register 16 Bytes	120h – 12Fh		1A0h – 1EFh
		Accesses 70h-7Fh	F0h – FFh	Accesses 70h-7Fh	130h – 16Fh 170h – 17Fh	Accesses 70h-7Fh	1F0h – 1FFh

Figura 10: Memoria de datos SRAM del PIC16F723A. Fuente:[1]

En la memoria de datos se distinguen dos tipos de registros:

- Registros de funciones especiales **SFR** (***S**pecial **F**uction **R**egisters*): Son los primeros registros de la memoria RAM y a través de ellos se controla parte de la funcionalidad del microcontrolador y se accede a sus diferentes periféricos, se programan sus funciones, etc.
- Registros de propósito general **GPR** (***G**eneral **P**urpose **R**egisters*): Se utilizan para guardar datos temporales y constituyen la memoria de programa disponible para el libre uso del usuario (192 bits totales).

Por lo tanto, el PIC16F723A implementa una memoria RAM de 128x4 bytes, es decir 4 bancos de memoria con 128 posiciones direccionables y de los cuales sólo 192 bytes son de tipo GPR para el almacenamiento de datos del usuario.

6.4. Procesador (CPU)

El PIC16F723A implementa un procesador de arquitectura HARVARD – RISC de 8 bits. La arquitectura HARVARD implica que el procesador se conecta a sus dos memorias (*SRAM* y *FLASH*) a través de buses independientes entre sí (Figura 11). Esto permite leer la próxima instrucción mientras se está ejecutando la instrucción actual.

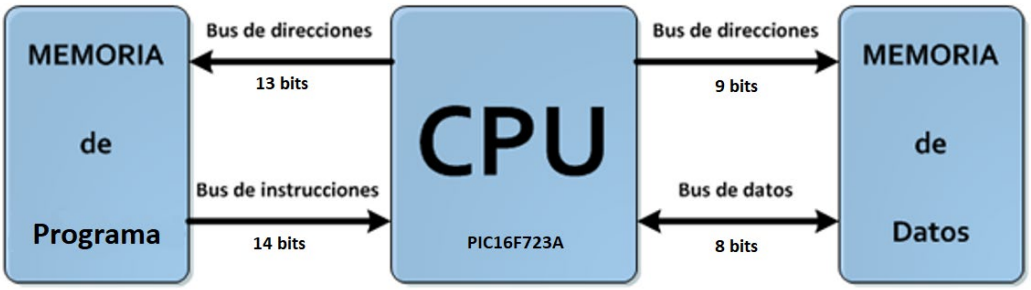


Figura 11: Diagrama de la arquitectura tipo HARVARD. Fuente: https://www.electrontools.com/Home/WP/2018/04/15/diferencias-entre-los-modelos-de-von-neumann-y-harvard/

Por otro lado, la arquitectura RISC (del inglés *Reduced Instruction Set Computer*) se caracteriza por realizar un diseño de las CPUs con un conjunto reducido de instrucciones y sencillo (En el caso del PIC16F723A solamente se implementan 35 instrucciones).

Este modelo de microprocesador también cuenta con una frecuencia del oscilador interno que incorpora (INTOSC) de hasta 16MHz, siempre y cuando se configure la opción de máxima frecuencia, dado que por defecto tras un reset trabaja a 8MHz. Teniendo en cuenta que en la familia de los PIC16, para ejecutar una instrucción, hacen falta 4 ciclos de reloj (obviando las instrucciones de salto las cuales necesitan 8 ciclos) este tipo de microcontrolador será capaz de ejecutar 4 millones de instrucciones por segundo.

6.5. Diagrama de pines del microcontrolador

Para ubicar los diferentes módulos que implementa el microcontrolador, a continuación, se muestra el diagrama de pines para el modelo PIC16F723A (Figura 12).

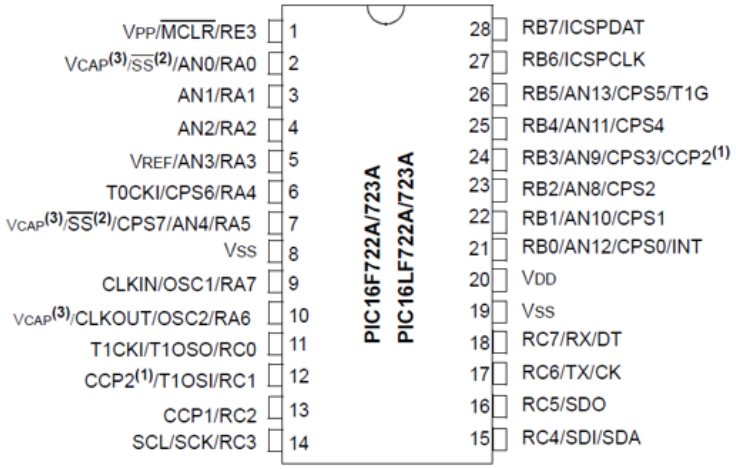


Figura 12: Diagrama de pines del PIC16F723A. Fuente:[1]

Observando la Figura 12 se aprecia que el PIC usado en el proyecto consta de 28 pines, los cuales tienen más de 28 funciones distintas. Esto es debido a que para optimizar espacio, internamente las conexiones con los pines están multiplexadas por uno o varios módulos. Por lo tanto, durante la programación del microcontrolador se deberá indicar la funcionalidad de cada pin a utilizar mediante la configuración que corresponda.

Para el funcionamiento del microcontrolador se requiere de la alimentación de los pines V_{DD} (Voltaje de alimentación) y Vss (Ground). El modelo utilizado puede funcionar con un rango de voltaje de alimentación entre 1,8V-5,5V.

6.6. Puertos de Entrada/Salida paralelos

Este microcontrolador implementa 4 puertos paralelos de Entrada/Salida, PORTA, PORTB, PORTC y PORTE. Un puerto (PORT) es una agrupación lógica de pines E/S direccionado en la memoria RAM de datos, por lo que su tamaño máximo será de 8 bits (1 byte). Además, cada registro de puerto (PORTx) lleva asociado un registro de dirección de datos (TRISx) que configura la dirección (entrada o salida) de cada uno de los pines del puerto (Figura 12).

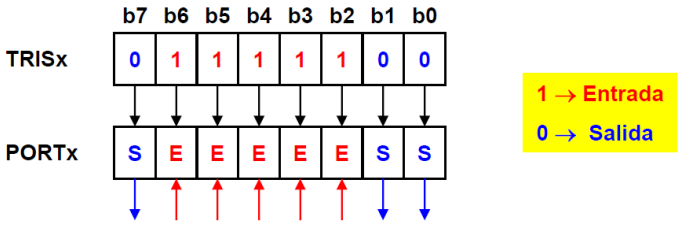


Figura 13: Esquema del funcionamiento de los puertos paralelos E/S. Fuente: Propia

Otro aspecto importante de los puertos E/S son sus características internas, es decir, cada puerto (o solo algunos pines) suelen tener características que los hacen diferentes al resto. En la Tabla 1 se pueden observar las características de los puertos del PIC16F723A.

I/O	28-Pin SPDIP, SOIC, SSOP	28-Pin QFN, UQFN	A/D	CCP	AUSART	SSP	Interrupt	Basic
RA0	2	27	AN0	—	—	$\overline{SS}$(3)	—	VCAP(4)
RA1	3	28	AN1	—	—	—	—	—
RA2	4	1	AN2	—	—	—	—	—
RA3	5	2	AN3/VREF	—	—	—	—	—
RA4	6	3	—	—	—	—	—	—
RA5	7	4	AN4	—	—	$\overline{SS}$(3)	—	VCAP(4)
RA6	10	7	—	—	—	—	—	OSC2/CLKOUT/VCAP(4)
RA7	9	6	—	—	—	—	—	OSC1/CLKIN
RB0	21	18	AN12	—	—	—	IOC/INT	—
RB1	22	19	AN10	—	—	—	IOC	—
RB2	23	20	AN8	—	—	—	IOC	—
RB3	24	21	AN9	CCP2(2)	—	—	IOC	—
RB4	25	22	AN11	—	—	—	IOC	—
RB5	26	23	AN13	—	—	—	IOC	—
RB6	27	24	—	—	—	—	IOC	ICSPCLK/ICDCLK
RB7	28	25	—	—	—	—	IOC	ICSPDAT/ICDDAT
RC0	11	8	—	—	—	—	—	—
RC1	12	9	—	CCP2(2)	—	—	—	—
RC2	13	10	—	CCP1	—	—	—	—
RC3	14	11	—	—	—	SCK/SCL	—	—
RC4	15	12	—	—	—	SDI/SDA	—	—
RC5	16	13	—	—	—	SDO	—	—
RC6	17	14	—	—	TX/CK	—	—	—
RC7	18	15	—	—	RX/DT	—	—	—
RE3	1	26	—	—	—	—	—	$\overline{MCLR}$/VPP

Tabla 1: Características (Reducida) de los puertos E/S paralelos. Fuente:[1]

De acuerdo a la Tabla 1, se listan a continuación algunos aspectos que se han considerado relevantes durante la realización del presente proyecto:

- Los puertos capaces de trabajar con señales analógicas son el PORTA y PORTB, aunque no en su totalidad de los pines.
- Los pines del PORTB, siempre y cuando estén configurados como entradas, pueden generar una solicitud de interrupción por cambio de flanco (de subida o de bajada).
- Los pines RC0 y RC1 del PORTC están conectados a los dos módulos CCP (Compare/Capture/PWM) que implementa el PIC, aunque este microcontrolador cuenta con la posibilidad de cambiar el pin CCP2 de RC1 a RB3.
- En el PORTC se encuentran los módulos de comunicación USART (Tx y Rx), I_2C (SDA y SCL) y SPI del inglés *Serial Peripheral Interface* (SDO, SDI y SCK).
- Tanto el PORTE como el PORTB sirven para programar el PIC mediante los pines RB6, RB7 y RE3 (ICSP del inglés *In-Circuit Serial Programming*).

6.7. Oscilador

Todos los microcontroladores requieren de un circuito interno o externo que les indique a que velocidad han de trabajar. A este tipo de circuitos se les conoce como osciladores de frecuencia y son utilizados para generar una onda cuadrada de alta frecuencia que configura los pulsos de reloj usados en la sincronización de todos los procesos del microcontrolador.

En el caso del PIC16F723A, el circuito reloj puede estar configurado por un reloj interno o externo. Además, en el caso del oscilador interno mediante diferentes multiplexores y un postscaler se obtiene una mayor flexibilidad respecto a las opciones de frecuencia de trabajo del microcontrolador. En la Figura 14 se muestra el esquema correspondiente a todos estos aspectos comentados.

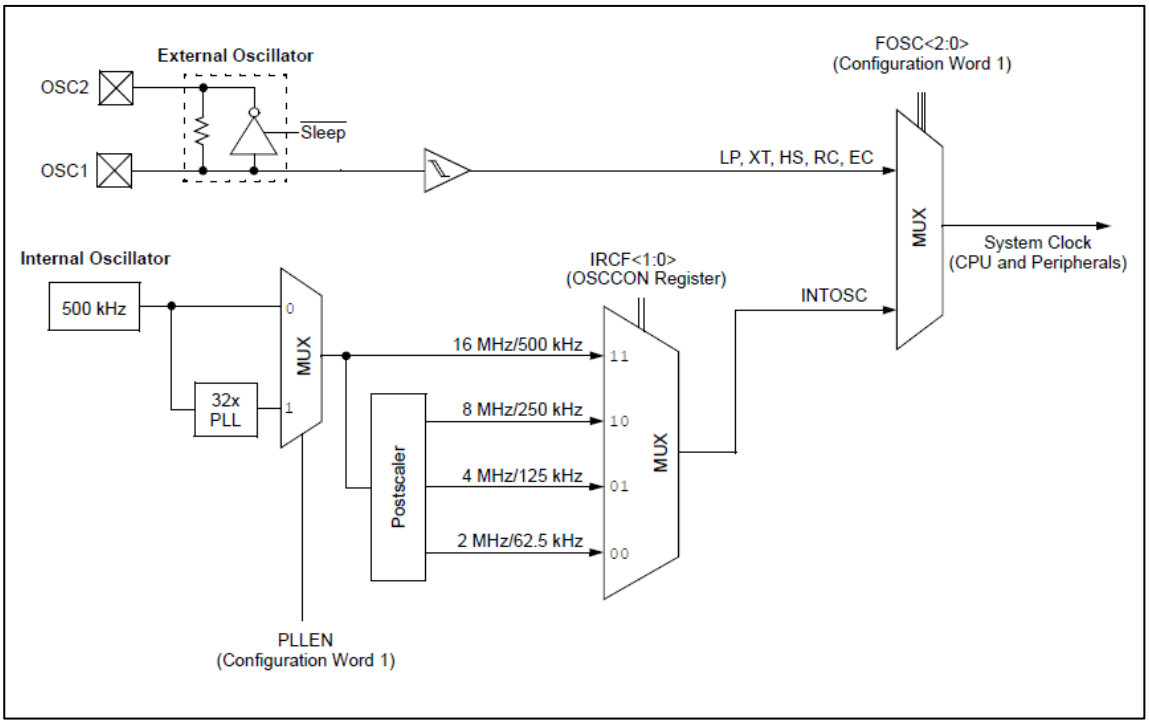


Figura 14: Diagrama de bloques simplificado de las fuentes de reloj del PIC16F723A. Fuente: [1]

En la configuración de frecuencia de la señal de reloj del PIC se utiliza el registro OSCCON (*Oscillator Control Register*), el cual permite la selección de las diferentes frecuencias del oscilador interno (INTOSC), como se detalla en la Figura 14.

U-0	U-0	R/W-1	R/W-0	R-q	R-q	U-0	U-0
—	—	IRCF1	IRCF0	ICSL	ICSS	—	—
bit 7							bit 0

bit 5-4 **IRCF<1:0>:** Internal Oscillator Frequency Select bits
When PLLEN = 1 (16 MHz INTOSC)
11 = 16 MHz
10 = 8 MHz (POR value)
01 = 4 MHz
00 = 2 MHz
When PLLEN = 0 (500 kHz INTOSC)
11 = 500 kHz
10 = 250 kHz (POR value)
01 = 125 kHz
00 = 62.5 kHz

Figura 15: Configuración de la frecuencia del INTOSC mediante el registro OSCCON. Fuente: [1]

Hay que tener muy en cuenta que tras un reset del microcontrolador, los bits IRCF<1:0> del registro OSCCON se configuran como '10' y la frecuencia pasa a ser 8MHz ó 250kHz.

6.8. Interrupciones

Las interrupciones realizadas sobre el programa principal son mecanismos muy utilizados para facilitar la conexión del sistema microprocesador con el exterior (periféricos), de esta forma, sincronizan la ejecución del programa principal con acontecimientos externos y facilitan la incorporación del microprocesador a sistemas que funcionan en tiempo real.

Su funcionamiento es muy parecido al de las subrutinas, pero se diferencian en la forma de puesta en marcha:

- Subrutina: Mecanismo software
- Interrupción: Mecanismo hardware

Pueden producirse en cualquier momento e implican la ejecución de la Rutina de Atención a la Interrupción (RAI), la cual se encuentra en la posición 0004h de la memoria de programa (Figura 9).

El funcionamiento del microprocesador al producirse una interrupción es el siguiente:

Al generarse una interrupción

- Se pone a '0' el bit GIE (*Global Interrupt Enable*) y de esta forma se impide que se atiendan dos interrupciones simultaneas.
- Se guarda la posición de retorno siguiente (PC) en la cima de la PILA.
- Se lleva al PC el vector de interrupciones (dirección 0x0004)

Cuando se retorna de una interrupción:

- Se activa el bit GIE para habilitar las interrupciones globales.
- Se transfiere la dirección de la cima de la PILA al PC.

Una vez se ha explicado como el funcionamiento en caso de darse una interrupción, a continuación, se listan todas las interrupciones que implementa el PIC16F723A:

- Interrupción externa por el terminal INT del microcontrolador (RB0)
- Interrupción por cambio en el nivel lógico de las entradas RB0:RB7 del Puerto B
- Interrupción por desbordamiento de los temporizadores TIMER0, TIMER1 y TIMER2
- Interrupción por algún evento en el módulo CCP
- Interrupción por el puerto serie AUSART
- Interrupción por el convertidor A/D

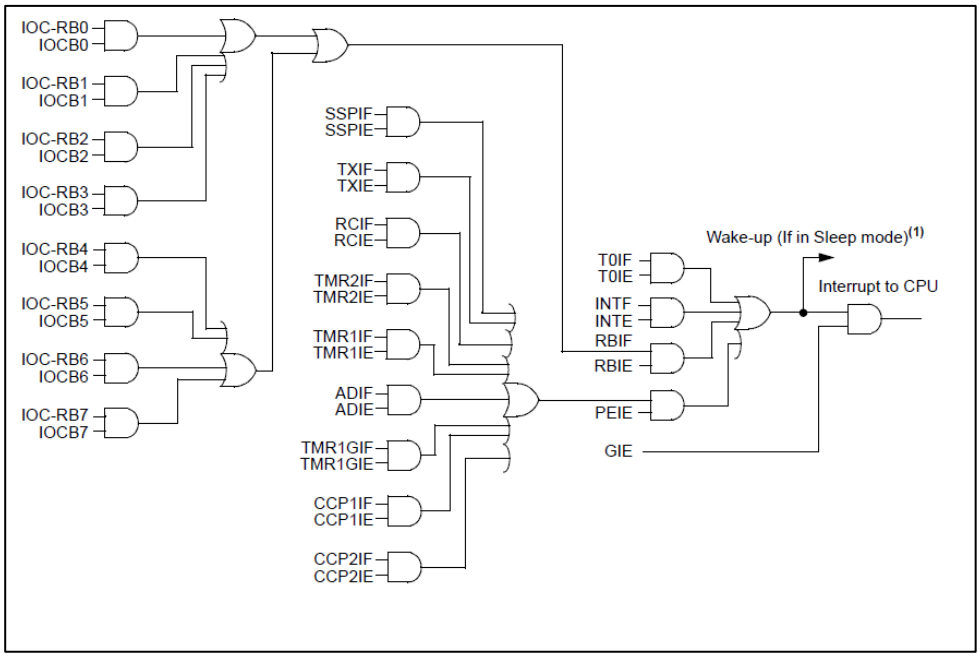


Figura 16: Lógica interna de las interrupciones del PIC16F723A. Fuente: [1]

En la Figura 16 se muestra el mapa lógico de interrupciones, el cual contiene tanto los bits de flancos de interrupción (terminación en F) como los bits que los habilitan (terminación en E).

Los registros asociados a estas interrupciones son los siguientes: ***INTCON***, ***PIE1***, ***PIE2***, ***PIR1***, ***PIR2***

Dirección	Registro	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0Bh, 8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
8Ch	PIE1	TMR1GIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
8Dh	PIE2	-	-	-	-	-	-	-	CCP2IE
0Ch	PIR1	TMR1GIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
0Dh	PIR2	-	-	-	-	-	-	-	CCP2IF

Figura 17: Registros asociados a las interrupciones del PIC16F723A. Fuente: Propia

En función de la interrupción que se quiera atender se tendrá que modificar o tener en cuenta los correspondientes registros mostrados en la Figura 17.

6.9. Temporizadores (TIMERS)

Los microprocesadores PIC de gama media incorporan hasta tres módulos básicos para temporizar, que se identifican con los nombres Timer0, Timer1, Timer2. Estos temporizadores son importantes cuando el microcontrolador debe trabajar con la variable tiempo. Por ejemplo, para generar señales de una determinada frecuencia, para medir la duración de una señal o simplemente para llevar la fecha y la hora con precisión.

Los temporizadores se diferencian entre sí por sus características internas tales como el contador síncrono de 8/16 bits, la posibilidad o no de contar pulsos externos, disponer de pre/post-divisor, disponer de opción de arranque/parada o reiniciarse automáticamente entre otras. A continuación, se muestra una tabla la cual recoge las características principales los Timer que implemente el PIC16F723A (Tabla 2)

Temporizador	Tamaño	Pre-divisor Divide entre:	Post-divisor Divide entre:	SFR donde está la cuenta	desbordamiento
Timer0	8 bits	2, 4, 8, 16, 32, 64, 128, 256	NO	TMR0	Bit T0IF de INTCON
Timer1	16 bits	1, 2, 4, 8	NO	TMR1H, TMR1L	Bit TMR1IF de PIR1
Timer2	8 bits	1, 4, 16	1, 2, … , 16	TMR2	Bit TMR2IF de PIR2

Tabla 2: Características de los TIMERS del PIC16F723A. Fuente: [1]

- **Timer 0**: Puede trabajar como timer de 8 bits, su pre-scaler esta compartido con el timer del Perro Guardián (en inglés *Watchdog*) y es posible configurarlo para contar pulsos externos. Se configura mediante el registro OPTION_REG (Figura 17).

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

Figura 18: Bits del registro OPTION_REG que configura el Timer 0. Fuente: [1]Tabla 2

Bit 7 ***RBPU***: Habilita/Deshabilita las resistencias internas de Pull-up del puerto B
1 = PORTB pull-ups deshabilitadas
0 = PORTB pull-ups habilitadas
Bit 6 ***INTEDG***: Selección del flanco para la generación de la interrupción externa por RB0
1=Interrupción externa RB0/INT flanco de subida
0=Interrupción externa RB0/INT flanco de bajada
Bit 5 ***T0CS***: Configura el Timer0 como temporizador (TC0S=0) o como contador (TC0S=1)
Bit 4 ***T0SE***: Configura el flanco de la señal externa con el que se incrementa el Timer0 si ha sido programado como contador
T0SE=0 para flanco de subida
T0SE=1 para flanco de bajada

Bit 3 ***PSA***: asigna el pre-divisor al Timer0 (PSA=0) o al perro guardián WDT (PSA=1)
Bit 2, 1 y 0 ***PS2:PS1:PS0***: Programan el factor de división del pre-divisor

PS2 PS1 PS0	Factor división para TMR0	Factor división para WDT
000	2	1
001	4	2
010	8	4
011	16	8
100	32	16
101	64	32
110	128	64
111	256	128

La temporización del Timer0 viene dada por la Ecuación 1:

$$Temporización\ Timer0 = (256 - N_{TMR0}) \cdot Prediv_{PS2:PS0} \cdot \frac{4}{f_{OSC}}$$

Ecuación 1: Temporización del TIMER0

Para una frecuencia de reloj de 16MHz, su temporización máxima es de **16,384 ms**.

- **Timer 1**: Para configurar la temporización del Timer1 se utiliza el registro T1CON (Figura 19).

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
TMR1CS1	TMR1CS0	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	—	TMR1ON
bit 7							bit 0

Figura 19: Bits del registro T1CON que configura el Timer1. Fuente: [1]

Bit 7-6 ***TMR1CS1:TMR1CS0***: Bits de selección de la fuente de reloj del TMR1

TMR1CS1-TMR1CS0	Fuente del CLK
00	Fosc/4
01	Fosc
10	Reloj externo (T1OSCEN=0) pin T1CKI Oscilador Cristal (T1OSCEN=1) pines T1OSI/T1OSO
11	Oscilador del sensado Capacitivo (CAPOSC)

Bit 5-4 ***T1CKPS1:T1CKPS0***: Bits de selección del factor de división del pre-divisor para la entrada de reloj Timer1

T1CKPS1-T1CKPS0	Factor de división
00	1
01	2
10	4
11	8

Bit 3 ***T1OSCEN***: Bit de habilitación (enable) del oscilador externo (T1OSC) del TMR1
1 = Oscilador habilitado
0 = Oscilador apagado
Bit 2 ***T1SYNC***: Bit de control de la sincronización del reloj externo del Timer1
Cuando *TMR1CS<1:0> = 1x*
1 = No sincroniza la entrada del reloj externo
0 = Sincroniza la entrada del reloj externo
Cuando *TMR1CS<1:0> = 0x*
Este bit se ignora. Dado que se usa el reloj interno que ya está sincronizado
Bit 0 ***TMR1ON***: Bit para el arranque/paro del Timer1
1 = Timer1 encendido (cuenta pulsos)
0 = Timer0 apagado

El cálculo para la temporización del Timer1 es el siguiente (Ecuación 2):

$$Temporización\ Timer1 = (65536 - N_{TMR1}) \cdot Prediv_{T1CKPS1:T1CKPS0} \cdot T_{CLK\ source}$$

Ecuación 2: Temporización del Timer1

La temporización máxima del Timer1 con frecuencia de trabajo del microcontrolador a 16 MHz y con fuente de reloj del Timer de $^{fosc}/_{4}$ es **131,072 ms**

- **Timer 2**: El Timer2 es un temporizador de 8 bits que cuenta con pre-scaler y post-scaler. Al contrario que el Timer0 y Timer1, el flanco de interrupción no se activa por desbordamiento sino por un comparador entre el registro TMR2 y el registro PR2 (Figura 20).

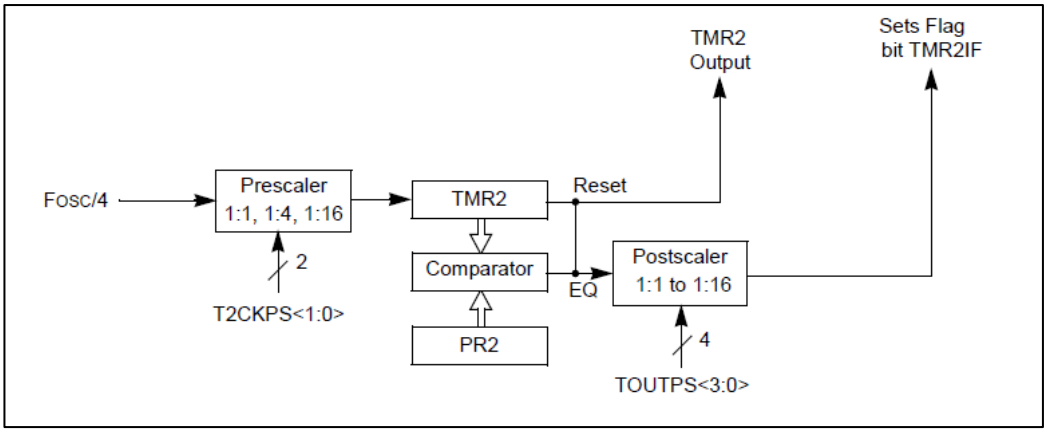


Figura 20: Diagrama de Bloques del Timer2. Fuente: [1]

El registro que se encarga de la configuración del Timer2 es el T2CON (Figura 21):

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

Figura 21: Bits del registro T2CON que configura el Timer2. Fuente: [1]

Bit 6-5-4-3 ***TOUTPS3:TOUTPS0***: Bits de selección del factor de división del post-divisor del Timer2

TOUTPS3-TOUTPS0	Factor de división
0000	1
0001	2
0010	3
0011	4
0100	5
0101	6
0110	7
0111	8
1000	9
1001	10
1010	11
1011	12
1100	13
1101	14
1110	15
1111	16

Bit 2 ***TMR2ON***: Bit de paro/arranque del TMR2
1 = Timer2 encendido (cuenta pulsos)
0 = Timer2 apagado
Bit 1-0 ***T2CKPS1:T2CKPS0***: Bits de selección del factor de división del pre-divisor del Timer2

T2CKPS1-T2CKPS0	Factor de división
00	1
01	2
1x	16

El cálculo de la temporización del Timer2 se calcula mediante la:

$$Temp\ Timer2 = Prediv_{TOUTPS3:TOUTPS0} \cdot (N_{PR2} + 1) \cdot Postdiv_{T2CKPS1:T2CKPS0} \cdot \frac{4}{f_{OSC}}$$

Ecuación 3: Temporización del Timer2

La temporización máxima que ofrece el Timer2 cuando el microcontrolador se encuentra trabajando a 16 MHz es de **16,384 ms.**

6.10. Módulos CCP en modo PWM

Los módulos CCP están formados básicamente por una pareja de registros de 8 bits denominados CCPRxH y CCPRxL y un registro CCPxCON de configuración (*Nota: Aquí y en lo sucesivo la letra "x" se ha de sustituir por 1 o 2 indicando el módulo CCP al que se esté haciendo referencia*).

En un mismo microcontrolador pueden existir hasta dos módulos CCP, denominados CCP1 y CCP2 y pueden operar en los siguientes modos:

- Modo de Captura: El módulo CCP captura el valor del Timer1 cuando ocurre un evento externo en el terminal CCPx.
- Modo Comparador: El registro del módulo CCP almacena un número de 16 bits que se compara con el valor del Timer1 y, según el resultado de la comparación, se genera un evento que puede incluir un cambio en el terminal CCPx.
- **Modo PWM (del inglés *Pulse Width Modulation*):** El módulo CCP y el Timer 2 forman un modulador de ancho de pulsos con salida por el terminal CCPx.

Dado que los módulos CCP comparten funciones con los temporizadores Timer 1 y Timer 2 en los PIC que disponen de dos módulos CCP, como sucede en el modelo PIC16F723A, hay que tener en cuenta el uso compartido de estos temporizadores para ambos módulos. Las posibles interacciones que hay que considerar a la hora de programar los módulos CCP se recogen a continuación (Tabla 3).

Modo del módulo CCPx	Modo del módulo CCPx	Interacción
Captura	Captura	Ambos módulos utilizan la misma base de tiempos (Timer 1)
Captura	Comparador	En el módulo CCP que opera como comparador debe configurarse para la puesta a 0 del Timer1 cuando el resultado de la comparación es positivo
Comparador	Comparador	Los comparadores deben configurarse para la puesta a 0 del Timer1 cuando el resultado de la comparación es positivo
PWM	**PWM**	**Ambas señales PWM tienen el mismo periodo, dado por el valor del registro PR2**
PWM	Captura	No hay interacción
PWM	Comparador	No hay interacción

Tabla 3: Interacción entre los módulos CCP en los distintos modos de operación para el PIC16F723A. Fuente: Propia

Una vez explicados los módulos CCP y vista la interacción que presentan con los Timers, dado que en el proyecto solo se utilizará la función PWM, se explicará este modo más en profundidad.

En el **modo PWM**, el objetivo es obtener a través de los terminales CCPx una señal de onda cuadrada como la mostrada en la Figura 22, donde tanto el periodo como el ciclo de trabajo (*Duty Cycle*) puede variar con el objetivo, por ejemplo, de regular la velocidad de un motor, la luminosidad de una bombilla, etc.

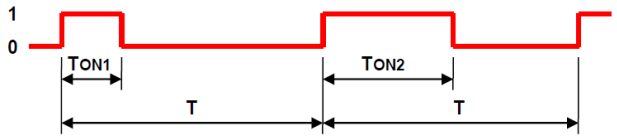


Figura 22: Forma de onda característica de una señal PWM. Fuente: Propia

Una señal PWM es un tren de pulsos periódicos de duración variable (T_{ON1}, T_{ON2}) y periodo T constante. Donde el ciclo de trabajo (*Duty Cycle*) se define como el porcentaje del periodo durante el cual la señal permanece en valor lógico alto.

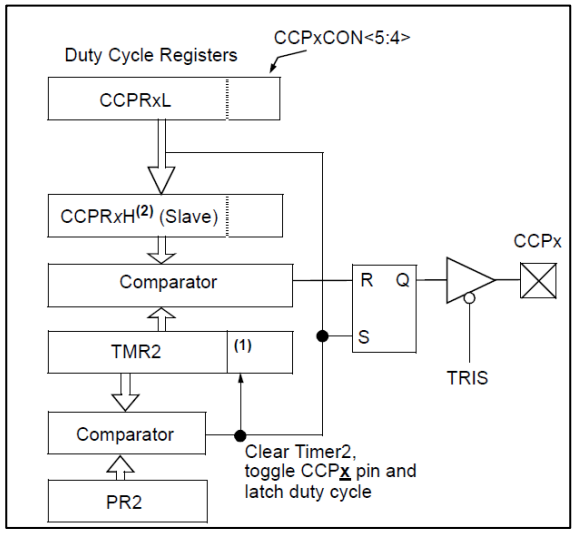


Figura 23: Diagrama de bloques simplificado del funcionamiento del modo PWM en el PIC16F723A. Fuente:[1]

Teniendo presente la Figura 23 y Figura 24, se procede a explicar el funcionamiento del módulo CCP en modo PWM:

- Se compara el valor almacenado en el registro PR2 con el valor cambiante de TMR2, de tal forma que, cuando ambos son iguales, significa que ha transcurrido el tiempo correspondiente a un periodo T de la señal PWM. Entonces se realizan las siguientes acciones:

- El registro TMR2 es puesto a 0 en el siguiente ciclo máquina.
- La entrada S del biestable RS se activa y con ello su salida Q pasa a 1. Por lo tanto, si el terminal CCPx está programado como salida mediante el bit correspondiente del registro TRIS, pasa también a 1.
- El valor situado en el *Duty Cycle Registers* (CCPRxL y CCPxCON<5:4>) se carga en el CCPRxH y en dos bits internos del módulo. Este valor es proporcional a la duración TON de los pulsos de la señal PWM.

➢ Realizadas estas acciones, se inicia un nuevo periodo de la señal PWM. El valor del *Duty Cycle Registers*, proporcional al ciclo de trabajo de la señal PWM, es comparado con el valor cambiante de TMR2 (más dos bits internos). Cuando ambos son iguales, la entrada R del biestable RS se activa y con ello la salida Q pasa a ser 0. Si el terminal CCPx ha sido programado como salida, pasa también a 0.

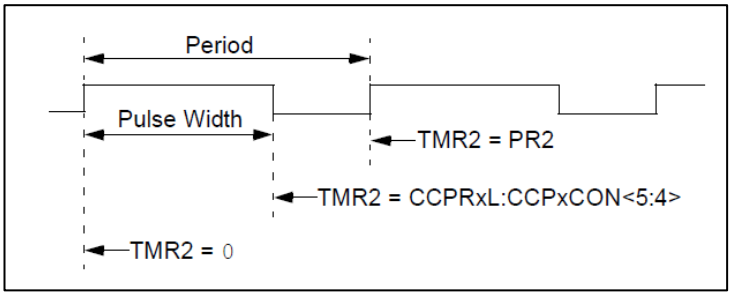


Figura 24: Funcionamiento del modo PWM del módulo CCP en el PIC16F723A. Fuente: [1]

Por lo tanto, para generar la PWM mediante el módulo CCP, se han de gobernar los parámetros, periodo (*PWM Period*) y ciclo de trabajo (*Duty Cycle Ratio*). Las ecuaciones de estos dos parámetros se muestran a continuación (Ecuación 4 y Ecuación 5).

$$PWM\ Period = [(PR2 + 1) \cdot 4 \cdot T_{OSC} \cdot TMR2_{PRESCALE\ VALUE}$$

Ecuación 4: Periodo de la señal PWM del módulo CCP. Fuente: [1]

$$Duty\ Cicle\ Ratio = \frac{(CCPRxL{:}CCPxCON\langle 5{:}4\rangle)}{4(PR2 + 1)}$$

Ecuación 5: Ciclo de trabajo de la señal PWM del módulo CCP. Fuente:[1]

6.11. Módulo AUSART

El módulo AUSART (del inglés *Addressable Universal Synchronous Asynchronous Receiver Transmitter*) es un periférico de comunicación que convierte los datos de formato paralelo a formato serie para ser transmitidos/recibidos de un nodo a otro. Estos tipos de circuitos son capaces de comunicarse tanto de forma asíncrona para comunicaciones Simplex, Half-Duplex y Full-Duplex, como síncrona para comunicaciones Simplex, siendo la forma asíncrona la que se utilizará en la comunicación *Bluetooth* del proyecto (*AUSART Asynchronous Mode*).

El módulo AUSART se puede dividir internamente en dos secciones bien diferenciadas, la de transmisión (Figura 24) y la de recepción (Figura 25).

- Sección Transmisora: El registro clave es el TSR (*Transmit Shift Register*), el cual no es totalmente direccionable mediante *software*. El TSR obtiene el dato a transmitir desde el registro TXREG, el cual actúa como buffer. La petición de interrupción se produce cuando el flag TXIF esté activo y habilitado (TXIE).

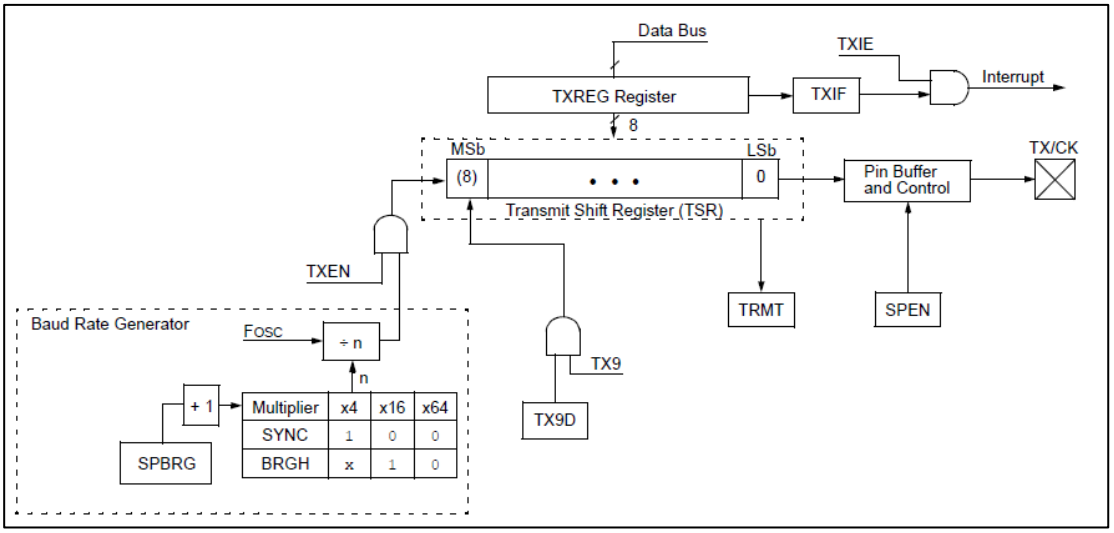


Figura 25: Diagrama de bloques de la sección Transmisora del AUSART (PIC16F723A). Fuente:[1]

- Sección Receptora: Los dos registros más importantes son el RSR (*Receive Shift Register*) y el FIFO (*First-In First-Out*). El "Data Recovery" opera a alta velocidad, mientras el RSR opera a velocidad normal de los bits. Una vez el RSR está configurado de forma correcta transfiere los datos al FIFO. Tanto el FIFO como el RSR no son direccionables directamente mediante *software*. La petición de interrupción se produce cuando el flag RCIF está activo y habilitado (RCIE).

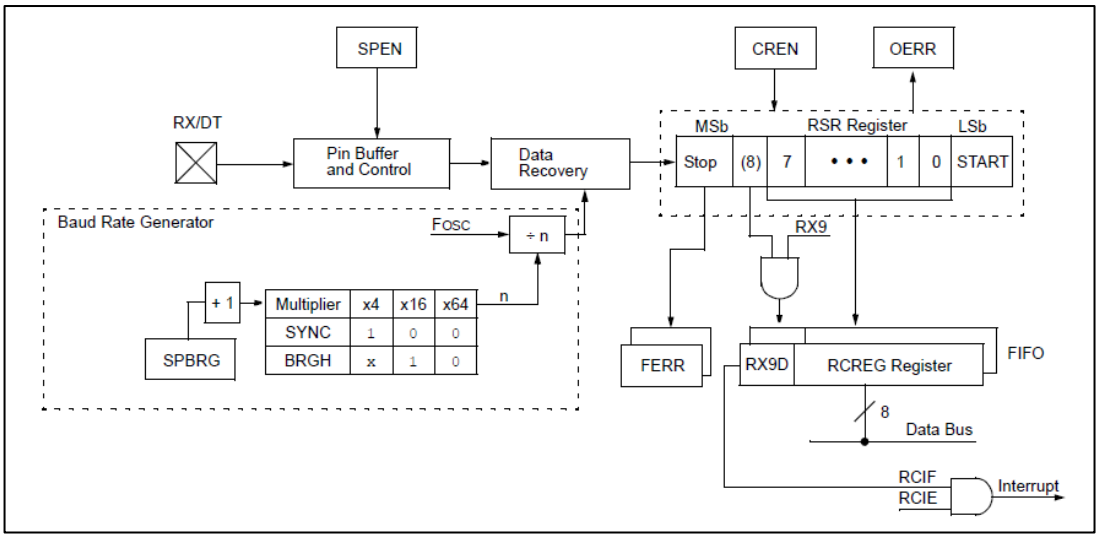


Figura 26: Diagrama de bloques de la sección Receptora del AUSART (PIC16F723A). Fuente:[1]

6.11.1. Configuración del módulo AUSART

Para realizar una comunicación de forma correcta entre ambos nodos, las secciones Transmisora y Receptora del módulo AUSART deben estar correctamente configuradas mediante los registros TXSTA (*Transmit Status and Control Register*) y RCSTA (*Receive Status and Control Register*), además de decidir la velocidad de transmisión mediante la configuración del BRG (***B**aud **R**ate **G**enerator*).

➢ Registro TXSTA (dirección 98h):

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN(1)	SYNC	—	BRGH	TRMT	TX9D
bit 7							bit 0

Figura 27: Bits del registro TXSTA del PIC16F723A. Fuente: [1]

Bit 7 ***CSRC***: Bit para la selección del reloj
Modo asíncrono
Se ignora
Modo síncrono
1 = Master (reloj interno
0 = Slave (reloj externo)
Bit 6 ***TX9***: Habilitación de la transferencia en modo 9 bits
1 = Transmisión 9 bits
0 = Transmisión 8 bits
Bit 5 ***TXEN***: Habilitación de la transmisión
1 = Transmisión habilitada
0 = Transmisión deshabilitada
Bit 4 ***SYNC***: Selección de modo
1 = Síncrono
0 = Asíncrono

Bit 2 ***BRGH***: Selección del modo de funcionamiento del generador interno
Modo asíncrono
1 = Alta velocidad
0 = Baja velocidad
Modo síncrono
Se ignora
Bit 1 ***TRMT***: Indicador de estado del registro de desplazamiento de transmisión
1 = Registro TSR vacío
0 = Registro TSR lleno
Bit 0 ***TX9D***: Noveno bit de transmisión de datos (puede ser el bit de paridad)

➢ Registro RCSTA (dirección 18h):

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

Figura 28: Bits del registro RCSTA del PIC16F723A. Fuente: [1]

Bit 7 ***SPEN***: Bit para la selección del reloj
1 = AUSART ON
0 = AUSART OFF
Bit 6 ***RX9***: Habilitación de la recepción en modo 9 bits
1 = Recepción 9 bits 0 = Recepción 8 bits
Bit 5 ***SREN***: Recepción de 1 bit
Modo asíncrono
Se ignora
Modo síncrono – Master
1 = Habilitada 0 = Deshabilitada
Modo síncrono – Slave
Se ignora
Bit 4 ***CREN***: Recepción continua
Modo asíncrono
1 = Habilitada 0 = Deshabilitada
Modo síncrono
1 = Habilitada hasta que CREN es 0
0 = Deshabilitada

Bit 3 ***ADDEN***: Detección de dirección
Modo asíncrono
1 = Permite detección de dirección, habilita la interrupción y la carga en el buffer de recepción cuando RSR<8> está a '1'.
0 = Deshabilita la detección de dirección, se reciben todos los bits y el bit '9' puede utilizarse como bit de paridad
Modo síncrono
Se pone a 0
Bit 2 ***FERR***: Indicador de error de trama
1 = Error
0 = Sin error
Bit 1 ***OERR***: Error por desbordamiento del registro de desplazamiento
1 = Error
0 = Sin error
Bit 0 ***RX9D***: Noveno bit de recepción de datos (puede ser el bit de paridad)

6.11.2. Recepción de datos en modo asíncrono del modo AUSART

La recepción en modo asíncrono de datos es el modo de comunicación más utilizado en el proyecto, por lo que se explicará el estado de los bits de los registros que intervienen en éste.

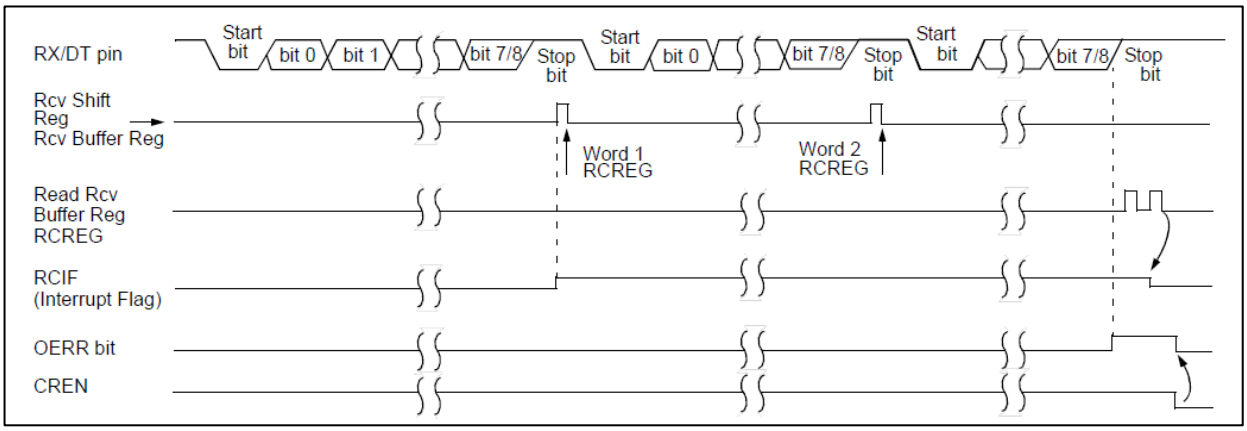


Figura 29: Ejemplo de comunicación asíncrona en el caso de recepción de datos. Fuente: [1]

Los aspectos más importantes observados en la Figura 29, se listan a continuación:

- En el pin RX se reciben los datos, de tal forma que al iniciar la transmisión desde el otro nodo (TX), se cambia el estado lógico de 5V a 0V durante un cierto periodo de tiempo creando así el bit de inicio. Una vez recibido el bit de inicio, se reciben los bits del dato, de menos a más significativo.

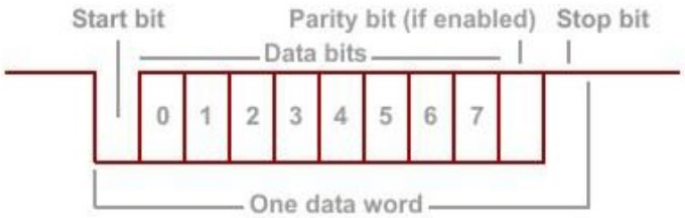


Figura 30: Patrón de funcionamiento de la lectura de datos (pin RX) del módulo USART.

- Una vez recibido los datos, el flanco de interrupción RCIF se activa, de esta forma indicará al microcontrolador que el dato ya ha sido leído y almacenado en el buffer de recepción RCREG. Al terminar la recepción, el RCIF es automáticamente puesto a '0'.

- Durante la recepción de datos, el bit CREN que indica la recepción continua de datos se mantiene en estado lógico alto.

6.11.3. Configuración del Baud Rate Generator (BRG)

El BRG (*Baud Rate Generator*) es un timer de 8 bits que se encarga de gestionar las comunicaciones síncronas y asíncronas del módulo AUSART.

Está formado por el registro SPBRG ("n" en la Tabla 4) el cual determina el periodo del temporizador de velocidad de transmisión. La velocidad de transmisión deseada en baudios depende del modo de comunicación a trabajar (asíncrono o síncrono) y de la velocidad configurada mediante el BRGH (alta o baja velocidad). Las diferentes opciones se muestran a continuación

Configuration Bits		AUSART Mode	Baud Rate Formula
SYNC	BRGH		
0	0	Asynchronous	FOSC/[64 (n+1)]
0	1	Asynchronous	FOSC/[16 (n+1)]
1	x	Synchronous	FOSC/[4 (n+1)]

Tabla 4: Fórmula de la velocidad de transmisión deseada en baudios del módulo AUSART. Fuente: [1]

7. Periféricos externos y conexiones

En este capítulo se explicará el funcionamiento de los diferentes periféricos y módulos externos utilizados en el proyecto, así como sus diferentes conexiones con el microcontrolador PIC utilizado.

7.1. Herramientas para la programación MPLAB IDE y PICkit3

En la programación del microcontrolador intervienen una serie de herramientas y programas informáticos que permiten generar y cargar en la memoria ROM el programa que controlará el PIC.

El *software* utilizado en la programación es MPLAB IDE v8.92 (Figura 31 a), el cual es el programa ensamblador/compilador que ofrece gratuitamente *Microchip Tecnology*. Se utiliza la versión 8.92 dado que es la última versión disponible de MPLAB IDE antes de que actualizaran al nuevo *software* MPLAB X. Para el compilador de lenguaje C, dado que esta versión no lo incluye en la instalación, se utiliza el *HI-TECH Universal ToolSuite* [2] (Figura 31 b).

a) b)

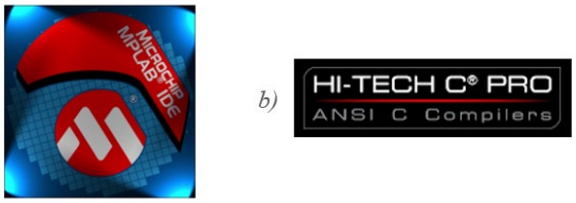


Figura 31: Software y compilador utilizados a) MPLAB IDE, b) HI-TECH C Compiler

El programador utilizado para cargar el programa realizado previamente en MPLAB IDE es el PICkit3 [3] (Figura 32), el cual se caracteriza por ser un *hardware* de bajo coste y no tener la opción de *Debugger* (permite hacer un seguimiento paso a paso de las líneas de código de programa que realiza el PIC, consultar el estado de sus registros internos, etc).

Figura 32: Hardware PICkit3 utilizado

El *software* MPLAB IDE, tal y como indica su nombre, es un editor IDE (*Integrated Developement Enviroment*) el cual permite la escritura de código en lenguaje ensamblador o lenguaje C, como en el caso de este proyecto (lenguaje de medio-alto nivel).

La programación en C permite que el programa realizado pueda funcionar con mínimas modificaciones para una gran variedad de microprocesadores con diferentes arquitecturas. Además, la generación de código C es bastante sencilla y facilita el entendimiento.

En la programación del microcontrolador PIC intervienen diferentes módulos descritos a continuación:

- Editor de Texto: Facilitan la escritura y limpieza del código mediante funciones como resaltado de sintaxis y sangría. Existen multitud de editores de texto, pero en este proyecto se utiliza el editor de texto incluido en el *software* MPLAB IDE con extensión de archivo .c

- Preprocesador: El preprocesador proporciona la capacidad de incluir archivos de encabezado (*header files, .h)*, expansiones de macros, compilación condicional y control de línea. Las directivas para el preprocesador han de empezar por #. Además, muchas veces este preprocesador es ejecutado durante la primera parte de la compilación.

- Compilador: Los compiladores son programas que traducen un fichero de código fuente de cualquier lenguaje a código máquina (lenguaje ensamblador). El objetivo de los compiladores es conseguir que el microcontrolador entienda el código ya que están diseñados para interpretar y procesar datos e instrucciones en forma binaria (‘0’ y ‘1’).

- Enlazador: Son los programas que enlazan varios ficheros objeto en lenguaje binario (*main program*, librerías, etc.) para crear una única imagen ejecutable del programa archivo con la extensión *.hex* que posteriormente se carga en la memoria del microcontrolador.

Para cargar el programa en el microcontrolador se utiliza el PICkit3. Este dispositivo se conecta con el ordenador vía un puerto USB y con el PIC mediante unos pines concretos destinados a la programación de éste (Figura 33).

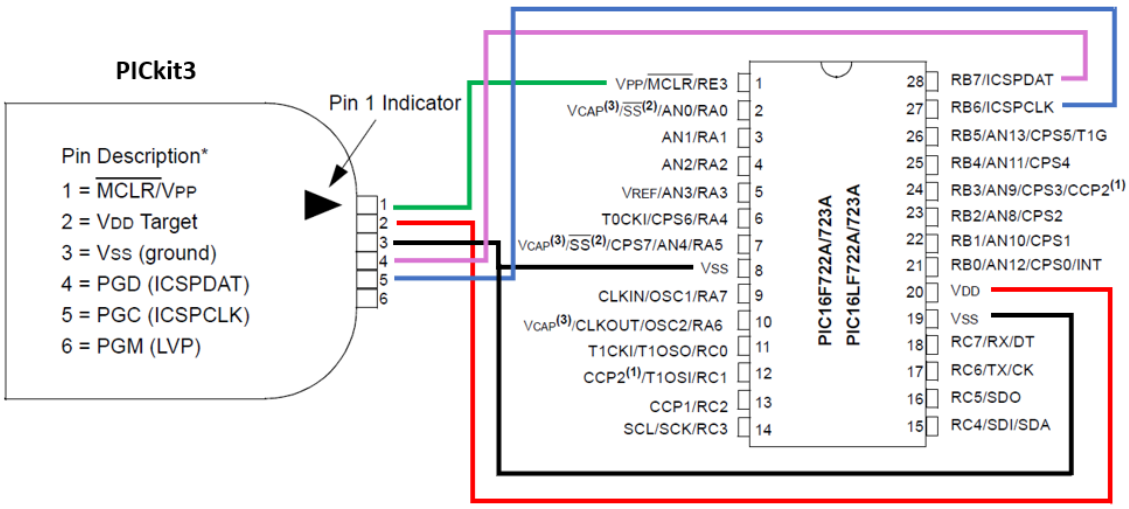


Figura 33: Conexiones del PICkit3 con el PIC16F723A. Fuente: Propia

7.2. Módulo Bluetooth HC-05

El módulo Bluetooth HC-05 permite establecer la comunicación Bluetooth entre el dispositivo *smartphone* del usuario y el microcontrolador para el envío/recepción de datos. Además, este modelo en concreto (HC-05) puede configurarse como Maestro (*Master*) o como Esclavo (*Slave*) y dispone de más funciones de más parámetros de configuración y capacidades de interrogación.

Figura 34: Módulo Bluetooth HC-05. Fuente: Propia

En la Figura 34 puede observarse los diferentes pines que implementa este módulo, siendo necesario una alimentación entre 3,6-6V y 0V de sus pines VCC y GND respectivamente.

Dispone también de otros 4 pines:

- Pin RX: Es utilizado en la recepción de información, siendo éste quien recibe los datos del nodo que transmite.
- Pin TX: Es utilizado en la transmisión de información, siendo éste quien envía los datos al nodo que los recibe.
- Pin Enable: Habilita/Deshabilita el funcionamiento del módulo Bluetooth (Normalmente no se conecta ya que en circuito abierto está activado, ENABLE).
- Pin State: Permite cambiar el módulo a modo de configuración. No está implementado físicamente en el pin, sino que se activa a través de mantener pulsado un botón de la cara superior durante un periodo de tiempo. El usuario en este modo podrá configurar diferentes parámetros del módulo mediante comandos AT(Figura 35).

COMANDOS AT (HC-05)

comando: AT <Enter> Respuesta OK //Test de comunicación.

comando: AT+NAME <Enter> Respuesta +NAME:HC-05 //Leemos su nombre.

comando: AT+NAME:NEW_NAME<Enter> Respuesta OK //Lo renombramos.

comando: AT+UART <Enter> Respuesta +UART:38400,0,0 //Leemos velocidad configurada.

comando: AT+UART:9600,0,0 <Enter> Respuesta OK //Reconfigurando velocidad.

comando: AT+ROLE <Enter> Respuesta +ROLE:0 //Leemos modo, en este caso esclavo.

comando: AT+ROLE:1 <Enter> Respuesta OK //Lo configuramos en modo master.

comando: AT+PSWD <Enter> Respuesta +PIN:"1234" //Leemos password actual.

comando: AT+PSWD:"0000" <Enter> Respuesta OK //Cambiamos password, lleva las comillas.

comando: AT+VERSION <Enter> Respuesta VERSION:2.0-20161226 //

Figura 35: Comandos AT para la configuración del módulo Bluetooth. Fuente: Propia

Las conexiones entre el módulo Bluetooth y el microcontrolador PIC son las siguientes:

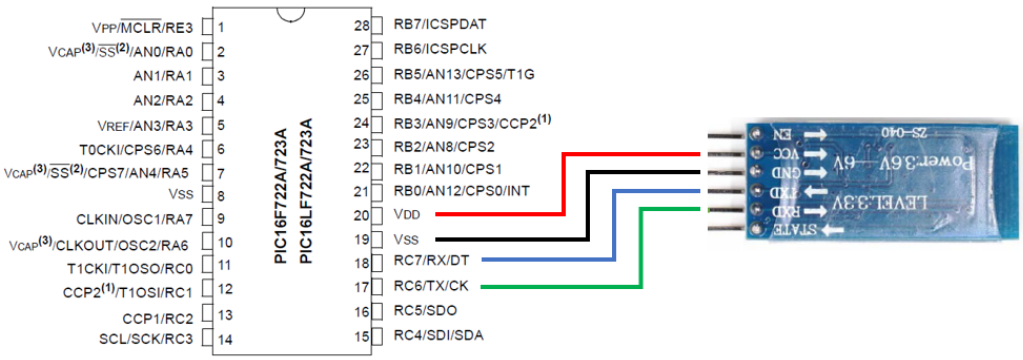


Figura 36: Conexiones del módulo Bluetooth HC-05 y el PIC16F723A. Fuente: Propia

7.3. Módulo L293D Mini Motor Shield

El módulo L293D (Figura 37) se encarga tanto de amplificar la corriente suministrada a los motores DC por medio de una batería externa, como de controlar y decidir el sentido de éstos.

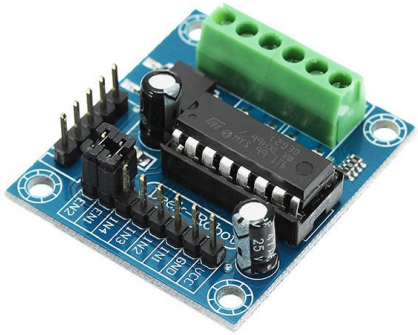


Figura 37: Motor Drive Shield Module Mini L293D. Fuente: Propia

Internamente cuenta con 4 circuitos amplificadores implementados mediante 4 buffers triestado (Figura 38), los cuales pueden usarse de manera independiente para controlar diferentes cargas y cada uno de ellos forma un medio puente en H (*half-bridge*). De esta forma mediante estos 4 circuitos pueden formarse 2 puentes en H completos (*full-bridge*), los cuales podrán controlar dos motores a través de sus salidas y de forma independiente entre sí.

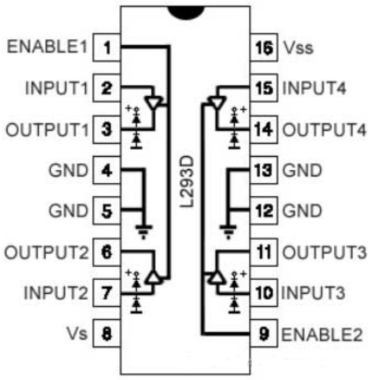


Figura 38: Pinout del módulo mini L293D. Fuente: https://www.microcontroller-project.com/l293d-pin-description-and-working.html#

Esta placa tiene dos partes de alimentación deparadas (Figura 39). Por un lado, se encuentra la parte de alimentación lógica (antes de la amplificación) con tensiones entre 0 y 5V en la cual se encuentran conectadas las salidas y alimentaciones del PIC y, por otro lado, la parte de alimentación de cargas (después de la amplificación) con valores entre 4,5V y 36V donde se conectan los motores y la alimentación de la batería.

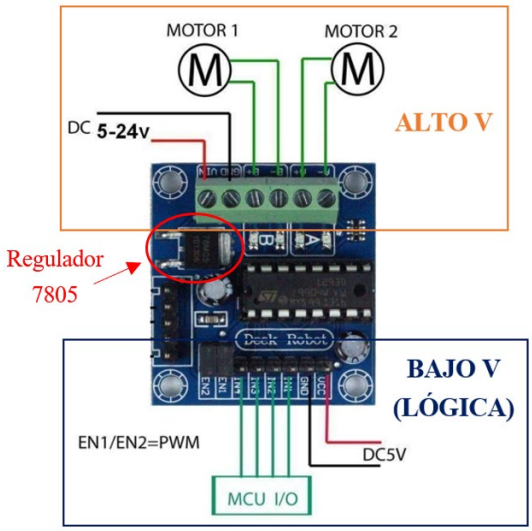


Figura 39: Diferentes Partes del módulo L293D. Fuente: Propia

Otra característica importante a tener en cuenta de este tipo de placa es el regulador 7805 con tecnología SMD que incorpora (Figura 39), el cual regula el voltaje de la batería conectada, a un voltaje estable de 5V alimentando tanto los motores DC (si ésta es la elección), como el PIC y todos sus periféricos. Además, permite que la corriente máxima de salida por canal para los motores DC sea de 600mA pudiendo alcanzar picos de hasta 1,2A [5].

Las conexiones de esta placa de amplificación con los motores DC y el PIC se muestran a continuación (Figura 40)

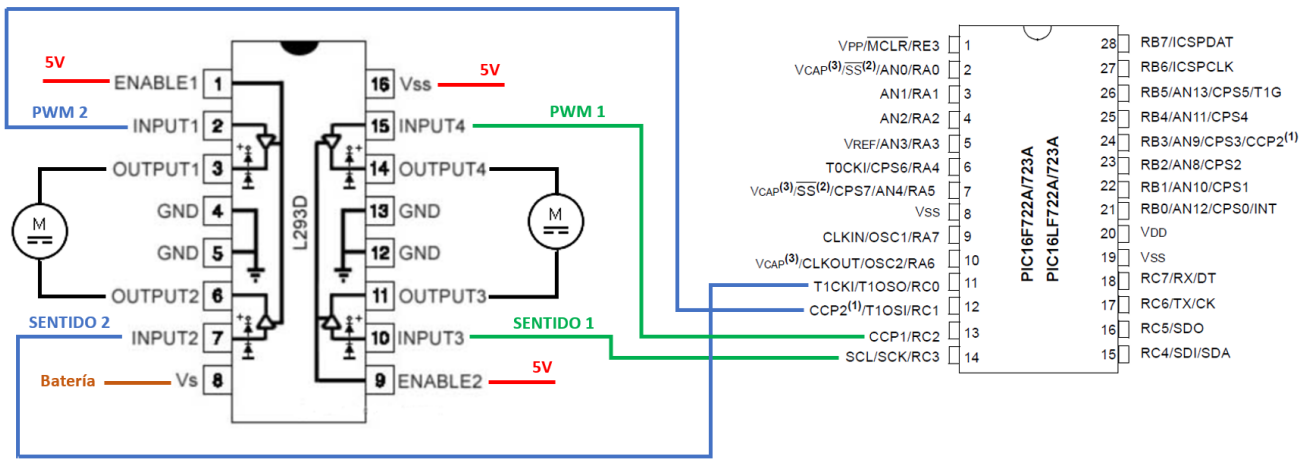


Figura 40: Conexiones del módulo L293D con el PIC y los motores DC. Fuente: Propia

En la alimentación de cada motor se usa un pin que tenga incorporada la función CCP configurado en modo PWM y otro pin I/O para determinar el sentido de giro.

7.4. Sensor de Ultrasonidos HC-SR04

El sensor de ultrasonidos tiene la función de detectar los obstáculos que se interponen en la trayectoria del vehículo, en el modo "Libre", y de esta manera evitar los posibles choques que se puedan producir.

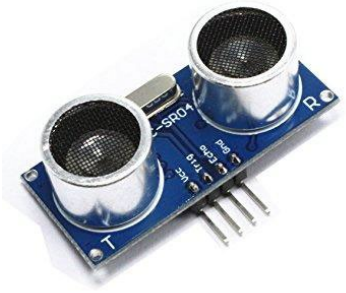


Figura 41: Sensor de Ultrasonidos HC-SR04. Fuente: https://www.330ohms.com/products/sensor-ultrasonico-hc-sr04

Los pines externos que implementa este sensor de ultrasonidos se listan a continuación:

- Vcc: Pin de alimentación del módulo a 5V
- Trigger: Pin de entrada para iniciar el disparo del ultrasonido
- Echo: Pin de salida que devuelve la señal rebotada en el objeto
- Gnd: Pin de alimentación del módulo a 0V

Una vez descritos los pines que implementa el sensor de ultrasonidos, se explica su funcionamiento. En primer lugar se genera un pulso de al menos 10us en la entrada "*trigger*", lo que provoca que se dispare una ráfaga de ultrasonidos de 8 ciclos a 40 kHz con el fin de detectar y rebotar en un objeto. Una vez rebotado la ráfaga en el correspondiente objeto, la salida "echo" devuelve un pulso de anchura proporcional al tiempo que ha tardado la ráfaga de ultrasonidos en ir del sensor al obstáculo y volver. El diagrama de tiempos del funcionamiento se muestra a continuación (Figura 42).

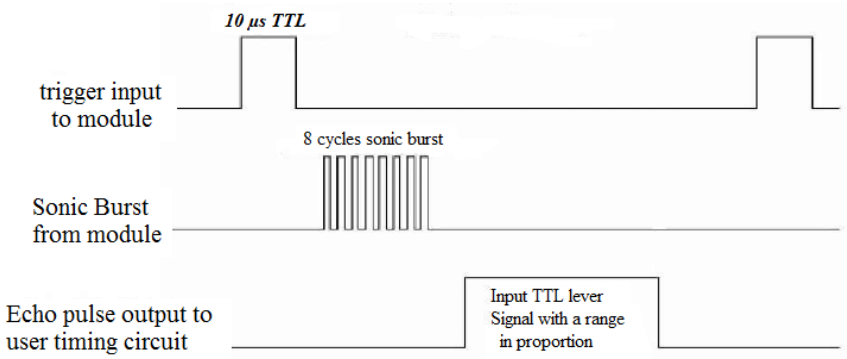


Figura 42: Diagrama de tiempo del funcionamiento del sensor de ultrasonidos HC-SR04. Fuente: [6]

El fabricante indica que tras realizar la medición de la duración del pulso de "*echo*", la distancia puede ser calculada a partir de la Ecuación 6:

$$Distancia(m) = Duración\ Pulso\ ECHO \cdot \frac{340\ m/s}{2}$$

Ecuación 6: Cálculo de la distancia (m) del Sensor Ultrasonidos. Fuente: [6]

Una vez explicado el funcionamiento y cálculo de la distancia, se realiza el esquema de conexiones entre el módulo y el microcontrolador PIC16F723A (Figura 43).

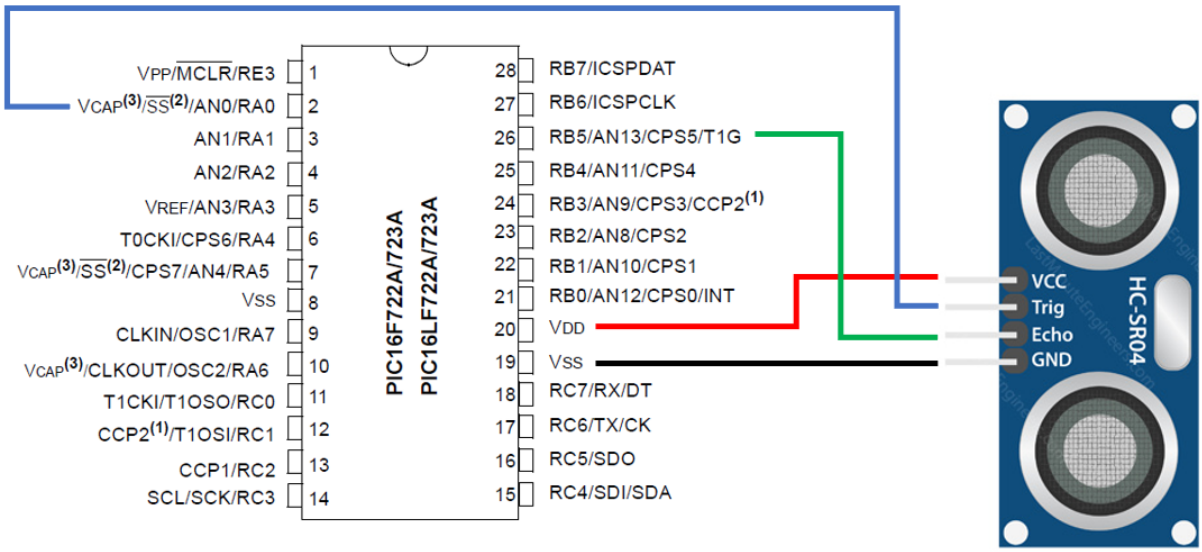


Figura 43: Conexiones del Sensor de Ultrasonidos con el PIC16F723A. Fuente: Propia

Respecto a la Figura 43, a la hora de realizar las conexiones para la salida "*echo"* del sensor de ultrasonidos se asigna un pin del PIC con capacidad de generar interrupciones de tipo IOC (*Interrupt On Change*), vistos en el apartado 6.6. Este tipo de interrupciones activan su flag cuando se detecta un cambio de flanco en la señal conectada, lo cuál permitirá detectar el inicio y final del pulso generado por el pin "*echo*". En este caso se utiliza el pin RB5 del PORTB.

8. Aplicación Android (App Inventor 2)

En este capítulo se explicará la aplicación desarrollada para realizar el control sobre el vehículo de 3 ruedas del proyecto.

El propósito de la aplicación se basa en establecer la comunicación entre el vehículo y el usuario, teniendo éste último la capacidad de decidir sobre los futuros movimientos a realizar por el coche.

El software utilizado para la creación de la app ha sido el software online *MIT App Inventor 2* creado por el MIT (*Massachusetts Institute of Technology*) el cual se caracteriza por ser de fácil uso y útil para realizar aplicaciones sencillas.

Figura 44: Software Online MIT App Inventor 2

El *App Inventor 2,* al igual que la mayoría de los entornos de desarrollo de aplicaciones móviles, cuenta con 3 herramientas diferentes:

- Gestor de Proyectos: Permite hacer un seguimiento de los proyectos realizados y visualizar diferentes características de éstos.
- Diseñador: Permite seleccionar los componentes de la app y definir la estructura visual para la interfaz de usuario.
- Editor de bloques: Es un espacio de trabajo donde se ensamblan los diferentes bloques o piezas del programa para especificar cómo deben comportase sus componentes.

8.1. Diagrama de Bloques General de la aplicación

Para tener una mejor visión de las diferentes pantallas que presenta la aplicación móvil, se realiza un diagrama de bloques con los diferentes procesos y opciones con los que tendrá que interactuar el usuario (Figura 45).

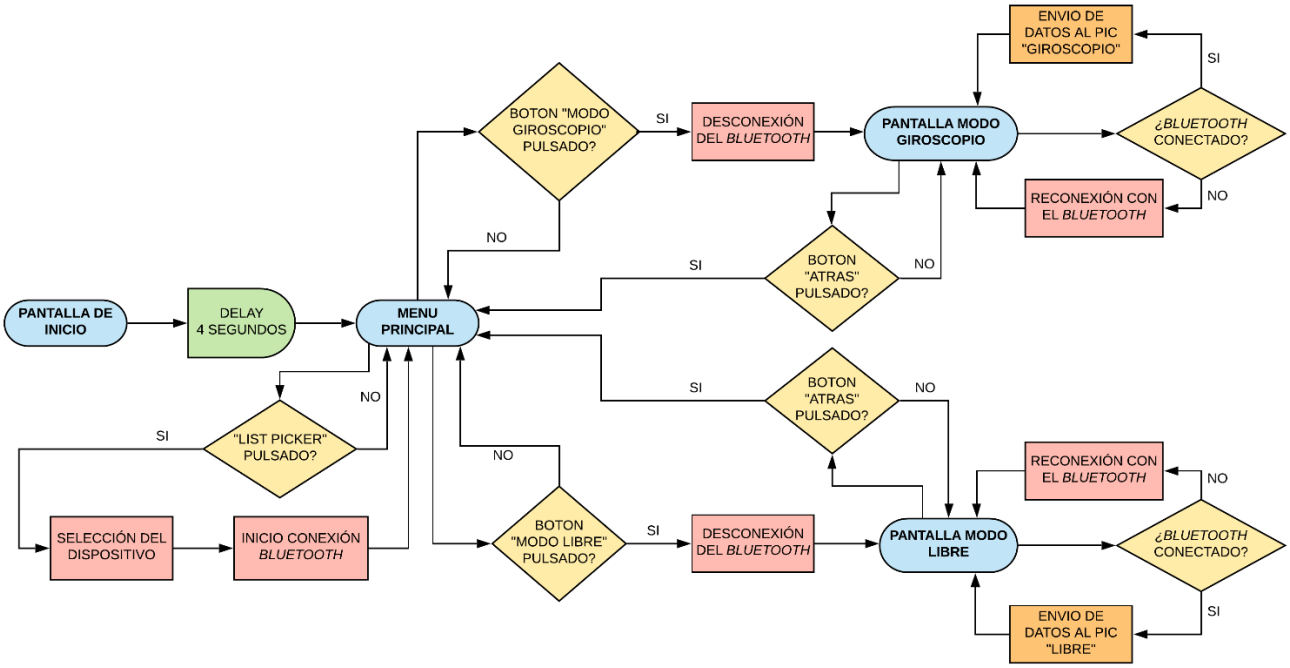


Figura 45: Diagrama de Bloques General de la aplicación móvil. Fuente: Propia

Como se puede observar en la Figura 45, la aplicación cuenta con 4 pantallas diferentes:

- Pantalla de Inicio: Es la primera pantalla de todas, cuyo objetivo es introducir el programa.
- Pantalla de Menú Principal: Pantalla principal que realiza la conexión *bluetooth* y la selección del modo de conducción.
- Pantalla Modo Giroscopio: Se realiza el control del vehículo a través de los valores del giroscopio del *smartphone*.
- Pantalla Modo Libre: Se realiza el control del vehículo a través de diferentes botones accionados por el usuario.

A continuación se detallan las características de cada una de las pantallas de la aplicación.

8.2. Pantalla de Inicio

El acceso a la Pantalla de Inicio se produce cuando se ejecuta la aplicación desde el teléfono *smartphone*. Su función es simplemente introductoria, es decir, presenta la aplicación del proyecto y una vez pasado un breve periodo de tiempo salta a la siguiente pantalla, Pantalla del Menú Principal.

8.2.1. Diseño

Para el diseño de la Pantalla de Inicio se han utilizado diferentes imágenes del vehículo y del logo de la universidad, así como la utilización de bloques de texto para presentar el título y autor del proyecto. Además, el uso de *layouts* ha permitido realizar un diseño visualmente ordenado y limpio. El resultado se muestra en la siguiente figura (Figura 46: Diseño de la Pantalla de Inicio. Fuente: PropiaFigura 46).

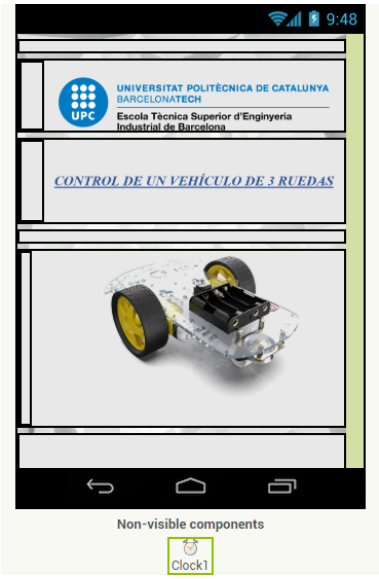


Figura 46: Diseño de la Pantalla de Inicio. Fuente: Propia

Por otro lado, en el diseño también se seleccionan y configuran los diferentes módulos para utilizar en cada pantalla, en este caso solo se utiliza un módulo “timer” configurado con temporización de 4000ms (Figura 47).

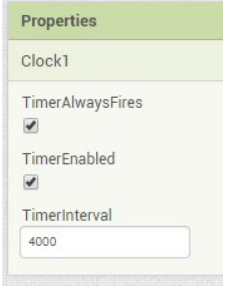


Figura 47: Configuración del módulo "clock" de la Pantalla de Inicio. Fuente: Propia

8.2.2. Programación

La programación de esta pantalla es la más sencilla de todas, dado que solamente implementa el módulo timer que pasado un tiempo en ms configurado (4000 ms) realiza un cambio a la siguiente pantalla.

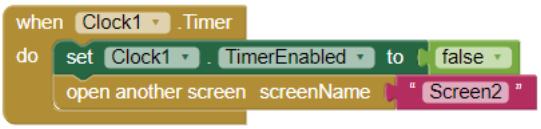


Figura 48: Programación por bloques de la Pantalla de Inicio. Fuente: Propia

Una vez realizada la temporización de los 4000ms se ejecuta la estructura de control "*when Clock1 Timer*" que deshabilita el timer y cambia a la pantalla "Screen2", la cual corresponde a la "Pantalla del Menú Principal".

8.2.3. Resultado

Una vez realizado la parte de Diseño y Programación de la Pantalla de Inicio, el resultado obtenido es el siguiente (Figura 48):

Figura 49: Resultado de la Pantalla de Inicio. Fuente: Propia

8.3. Pantalla de Menú Principal

La Pantalla de Menú Principal es la interfaz más importante de la aplicación móvil, dado que permite al usuario acceder a los diferentes modos de conducción e iniciar/finalizar la comunicación vía *Bluetooth* con el vehículo.

8.3.1. Diseño

En el diseño de esta pantalla se utilizan 3 botones y 1 *list-picker* (botón que despliega una lista) que actúan como inputs accionados por el usuario. Dos de los botones sirven para seleccionar entre los modos, "Modo Giroscopio" y "Modo Libre", y el último de ellos se utiliza para finalizar la conexión *Bluetooth* entre móvil y vehículo. Por otro lado, la *list-picker* permite visualizar los dispositivos *Bluetooth* detectados por el móvil y de esta forma iniciar la comunicación con el vehículo. Finalmente, se utilizan diferentes *layouts* para organizar el diseño de la pantalla y un cuadro de texto que muestra el nombre del dispositivo conectado (Figura 50).

Figura 50: Diseño de la Pantalla de Menú Principal. Fuente: Propia

Los módulos seleccionados en esta pantalla son el módulo *Bluetooth*, un módulo de almacenamiento para guardar el nombre del dispositivo conectado (necesario para la conexión *Bluetooth* multi-pantalla) y un módulo de notificaciones.

8.3.2. Programación

La programación de esta pantalla es una de la más compleja de todas, internamente se puede dividir en 3 partes diferentes, la primera correspondiente a la conexión/desconexión de la comunicación *Bluetooth* realizada por un botón y una *list-picker*, la segunda parte engloba el cambio a los modos "Libre" Y "Giroscopio" mediante 2 botones diferentes y, finalmente, la tercera parte cuyo objetivo es reconectar la comunicación *Bluetooth* al regresar de una pantalla diferente.

La programación por bloques de toda la pantalla y sus diferentes partes, se muestran a continuación (Figura 51).

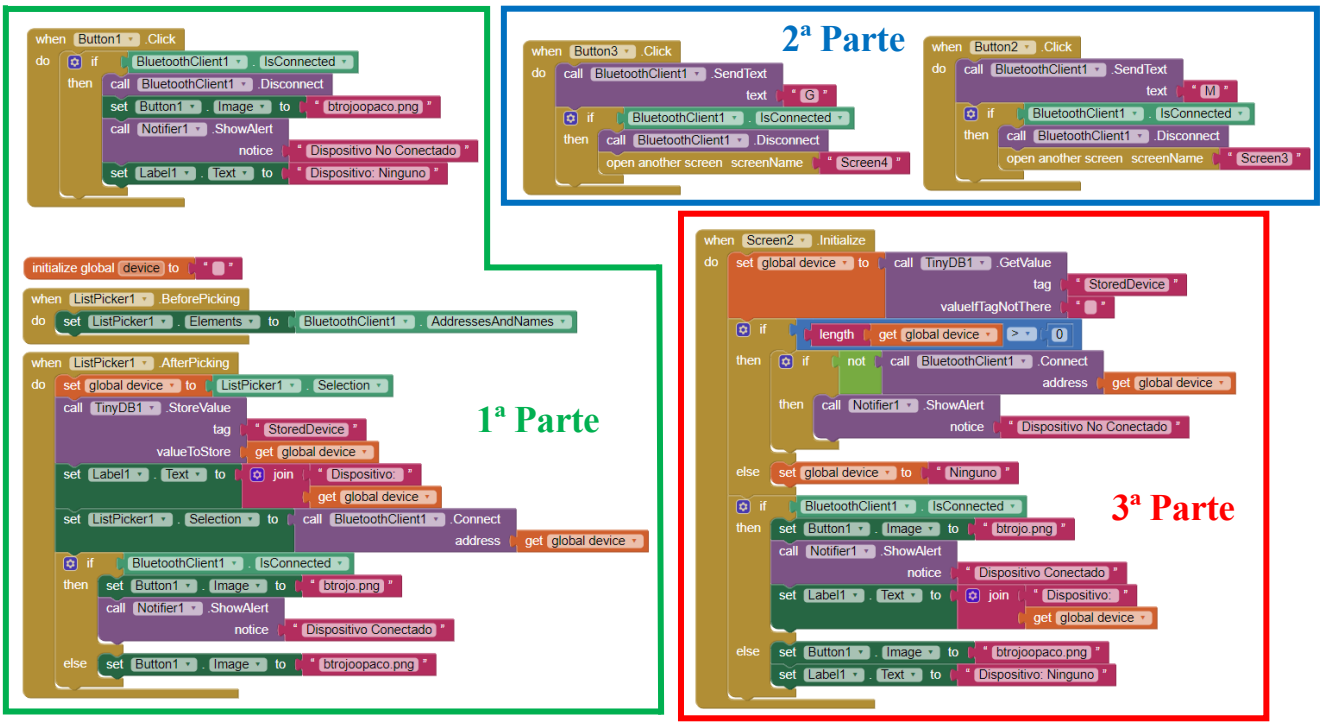


Figura 51: Programación de la Pantalla de Menú Principal. Fuente: Propia

Para entender mejor en funcionamiento de los diferentes bloques, se explican estas tres partes por separado.

8.3.2.1. Conexión/Desconexión *Bluetooth*

Esta primera parte permite el accionamiento de 2 inputs diferentes por el usuario que permitirán la Conexión/Desconexión de la comunicación *Bluetooth* entre el móvil y el vehículo.

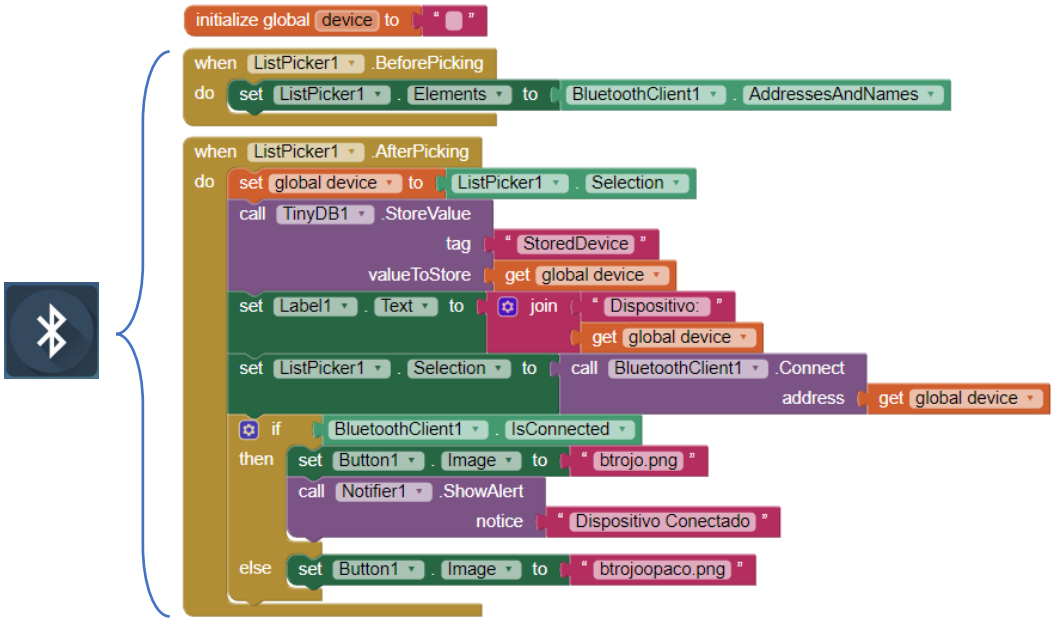


Figura 52: Programación por Bloques de la list-picker (Menú Principal). Fuente: Propia

La Figura 52, muestra las acciones realizadas por la *list-picker* para iniciar la comunicación *BT*. Las acciones que se realizan se listan a continuación en su orden correspondiente:

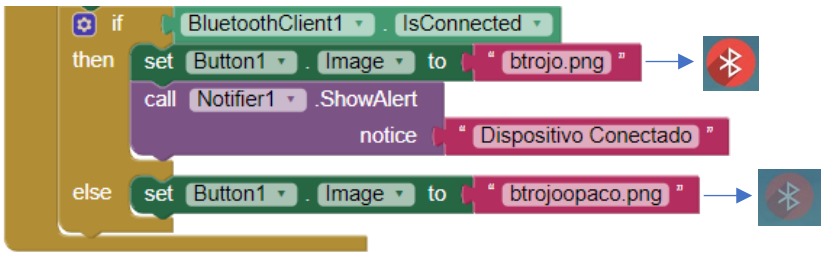
- Se inicia la variable "device" de tipo global, la cual almacenará el nombre del dispositivo.

- Una vez se acciona la *list-picker* se visualizan las diferentes direcciones y nombres de los dispositivos a conectar.

- Se selecciona el dispositivo correspondiente al vehículo a controlar, guardándose tanto en la variable "device" creada anteriormente como en el módulo de almacenamiento "TinyDB1". De esta forma se conocerá la dirección y nombre del dispositivo seleccionado en todas las pantallas que implementen este módulo de almacenamiento y de esta forma poder reconectar la conexión *Bluetooth* con el dispositivo.

- Se inicia la conexión *Bluetooth* teléfono/vehículo además de mostrar en la pantalla mediante un cuadro de texto que dispositivo se ha conectado.

- En el caso de que el inicio de la conexión se haya realizado correctamente y el dispositivo se encuentre conectado, se cambia el botón de desconexión de color opaco (inhabilitado) a color rojo (habilitado) y muestra una notificación de "Dispositivo Conectado". En caso contrario, el botón de desconexión se mantiene en color opaco.



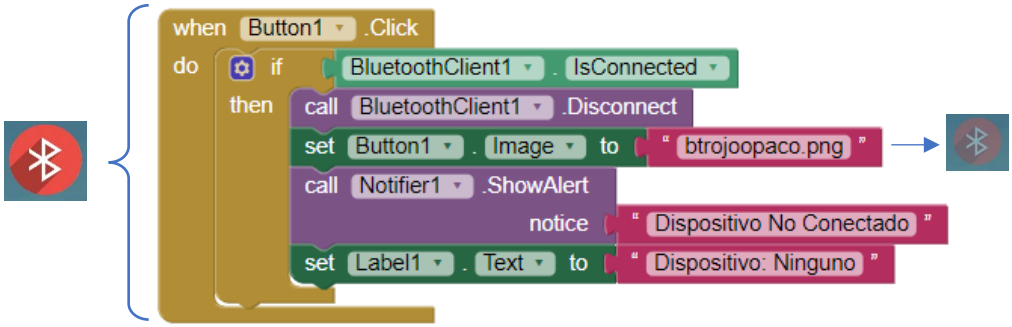


Figura 53: Programación por Bloques del botón de Desconexión BT. Fuente: Propia

Para finalizar la conexión *Bluetooth* sólo se ha de pulsar el botón de desconexión. En caso de estar conectado el dispositivo, finaliza esta conexión y cambia el botón a color opaco, además muestra una notificación de "No Conectado" y el cuadro de texto que muestra el nombre del dispositivo pasa a mostrar "Ninguno" (Figura 53).

8.3.2.2. Selección de los modos "Libre" y "Giroscopio"

Para permitir al usuario acceder a los modos "Libre" y "Giroscopio", se dispone de dos botones diferentes, cuya programación es similar salvo los datos que se envían al vehículo (Figura 54).

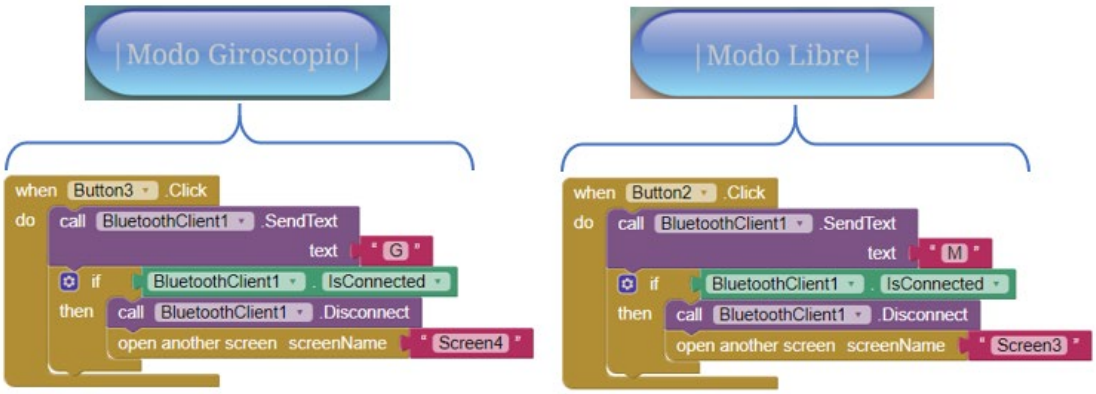


Figura 54: Programación por Bloques de los botones modo "Giroscopio" y modo "Libre". Fuente: Propia

Al realizar la pulsación de cualquiera de los botones se envía un Byte de datos en código ASCII (cada letra tiene un patrón de 8 bits diferente) y es recibido por el módulo *Bluetooth* del vehículo. En el caso del modo "Giroscopio" se envía la letra "G" y en el caso del modo "Libre" la letra "M".

Finalmente, en caso de que el *Bluetooth* esté conectado, se desconecta y es entonces cuando se cambia de pantalla. Esta desconexión previa al cambio de pantalla tiene como objetivo el correcto funcionamiento de la reconexión *Bluetooth* multi-pantalla.

8.3.2.3. Reconexión de la comunicación *Bluetooth*

La reconexión de la comunicación *Bluetooth* se realiza de forma automática cada vez que se inicializa la Pantalla de Menú Principal. De esta forma al retornar de otra pantalla el dispositivo *smartphone* seguirá conectado con el vehículo sin necesidad de que el usuario vuelva a realizar el proceso de conexión BT.

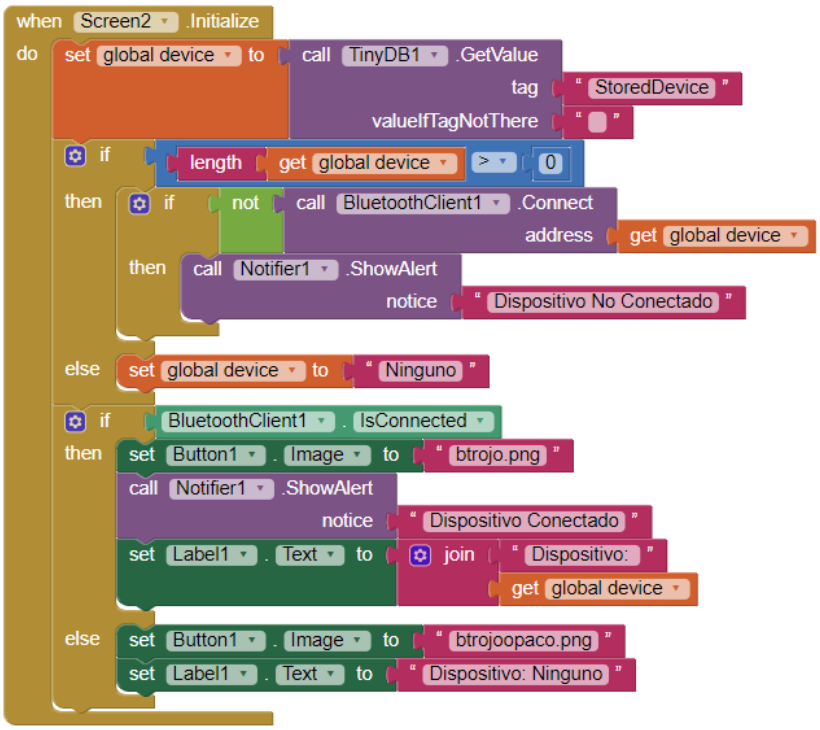
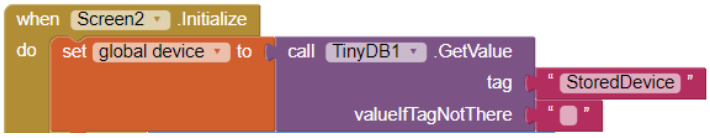


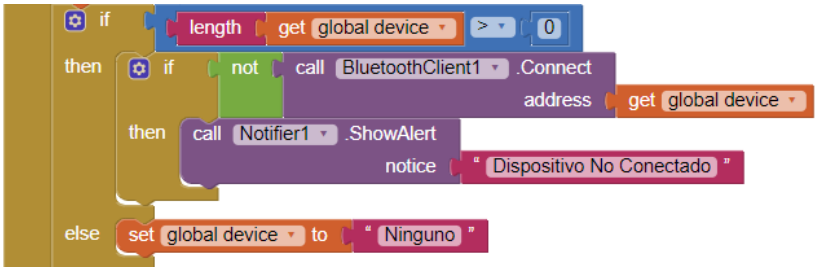
Figura 55: Programación por Bloques de la reconexión BT en el Menú Principal. Fuente: Propia

A continuación, observando la Figura 55 se listan los diferentes pasos que se realizan en esta reconexión de la comunicación *Bluetooth.*

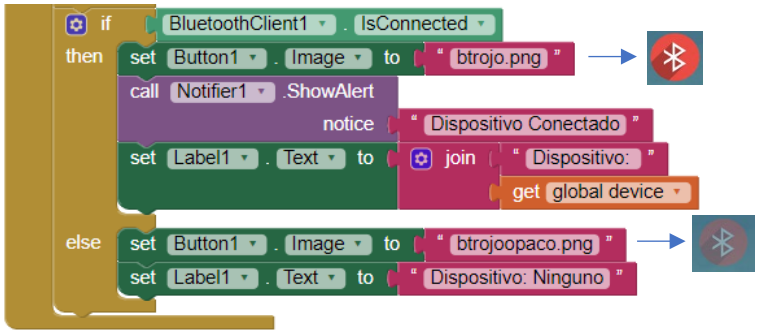
- Una vez iniciada la pantalla se retorna del módulo de almacenamiento "TinyDB1" la dirección y nombre del dispositivo guardado durante el proceso de conexión de la comunicación BT (8.3.2.1).



- Se comprueba si realmente había un dispositivo guardado anteriormente, en caso de no haberlo el nombre del dispositivo se establece en "Ninguno". Por el contrario, en caso de sí haberlo, se realiza la reconexión *Bluetooth* con este dispositivo y una vez terminada, se comprueba si el proceso de conexión se ha realizado de forma correcta, en caso de no ser así se muestra un mensaje de notificación "Dispositivo No conectado"



- Finalmente, si la comunicación *Bluetooth* se encuentra conectada, se habilita el botón de desconexión BT, se muestra una notificación de "Dispositivo Conectado" y el cuadro de texto es actualizado para mostrar la dirección y nombre del dispositivo conectado. En caso contrario, el botón de desconexión se deshabilita y el cuadro de texto muestra "Ninguno".



8.3.3. Resultado

Una vez realizados tanto el apartado de diseño como el de programación, el resultado obtenido es el siguiente.

- Con dispositivo No Conectado: Se muestra la notificación "Dispositivo No Conectado", el nombre del dispositivo "Ninguno" y el botón de desconexión BT deshabilitado.

Figura 56: Resultado con Dispositivo No Conectado de la Pantalla de Menú Principal. Fuente: Propia

- Con dispositivo Conectado: Se muestra la notificación "Dispositivo Conectado", la dirección y nombre del dispositivo conectado y el botón de desconexión BT habilitado.

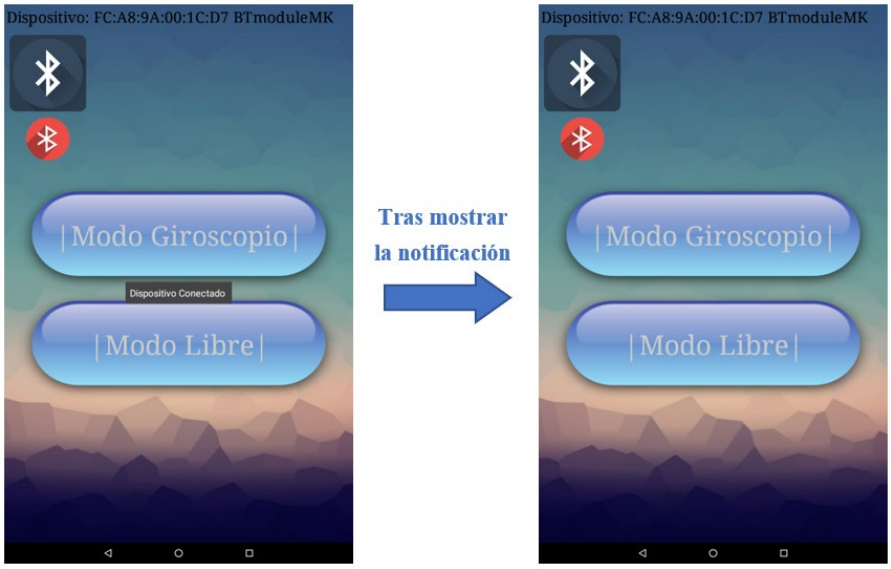


Figura 57: Resultado con Dispositivo Conectado de la Pantalla de Menú Principal. Fuente: Propia

8.4. Pantalla del "Modo Giroscopio"

La Pantalla del Modo Giroscopio, permite realizar el control del vehículo mediante el giroscopio de tipo MEMS (Sistemas Micro Electro Mecánicos) que incorpora el dispositivo *smartphone* y la comunicación *Bluetooth* previamente configurada en el Menú Principal.

8.4.1. Diseño

En el diseño de esta pantalla (Figura 54) se utilizan diferentes *Sprites* (imágenes con capacidad para detectar colisiones) contenidos en un *Canvas* (panel rectangular animado que contiene los sprites y puede moverlos). Además, también se utilizan dos cuadros de textos que muestran los datos de enviados de los ángulos "Roll" y "Pitch" con 2 decimales.

Figura 58: Diseño de la Pantalla del Modo Giroscopio. Fuente: Propia

Algunos de los módulos que se han seleccionado para esta pantalla son los mismo que en la Pantalla de Menú Principal (8.3.1) para todo lo relacionado con la comunicación *Bluetooth*. Por otro lado, también incluye el módulo giroscopio del móvil (*Orientation Sensor*) que devuelve los valores Roll/Pitch/Yaw y un módulo temporizador configurado a 250ms (Figura 59) para no saturar el envío de datos del móvil al vehículo.

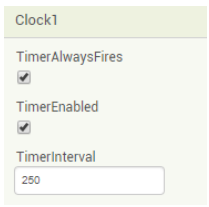


Figura 59: Configuración del módulo "clock" de la Pantalla del Modo Giroscopio. Fuente: Propia

8.4.2. Programación

La programación de la Pantalla del Modo Giroscopio engloba tanto el cálculo y envío de los datos Roll y Pitch al PIC, la reconexión de la comunicación *Bluetooth* multi-pantalla, el retorno mediante el botón "*back*" al Menú Principal y la animación de los *sprites* del canvas que el usuario podrá visualizar.

8.4.2.1. Reconexión de la comunicación *Bluetooth*

Esta parte de la programación es idéntica que la anteriormente explicada en la Pantalla de Menú Principal (8.3.2.3), a diferencia de que, al no tener botón de desconexión de la comunicación *Bluetooth,* esa parte no está presente (Figura 60).

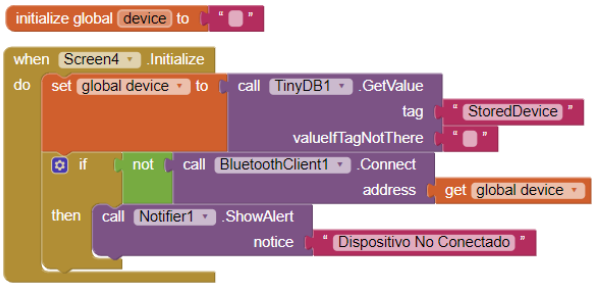


Figura 60: Programación por Bloques de la reconexión BT en la Pantalla del Modo Giroscopio. Fuente: Propia

8.4.2.2. Animación de los *Sprites* del Canvas

En este apartado de la programación se aplica las diferentes animaciones a los *sprites* del canvas a partir de las colisiones entre ellos y los ángulos Roll/Pitch obtenidos por el módulo giroscopio tipo MEMS.

- Control del movimiento del *sprite* del vehículo: Para aplicar al *sprite* un movimiento de manera correcta, se realizan diferentes cálculos que evitan vibraciones y movimientos bruscos de este (Figura 61).

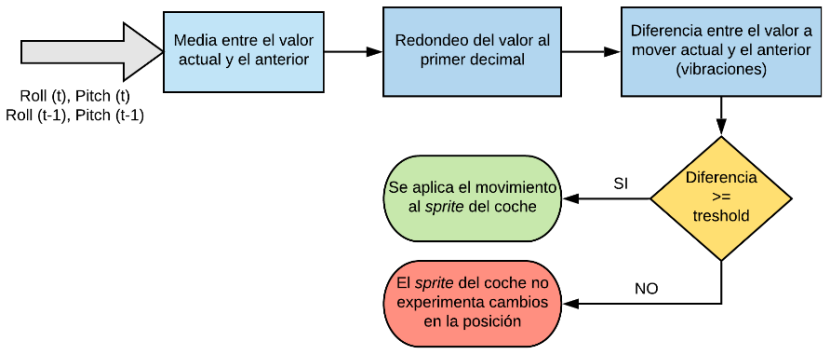


Figura 61: Diagrama de Bloques del movimiento del sprite del vehículo. Fuente: Propia

A continuación, en la Figura 62 se muestra la programación por bloques de esta parte.

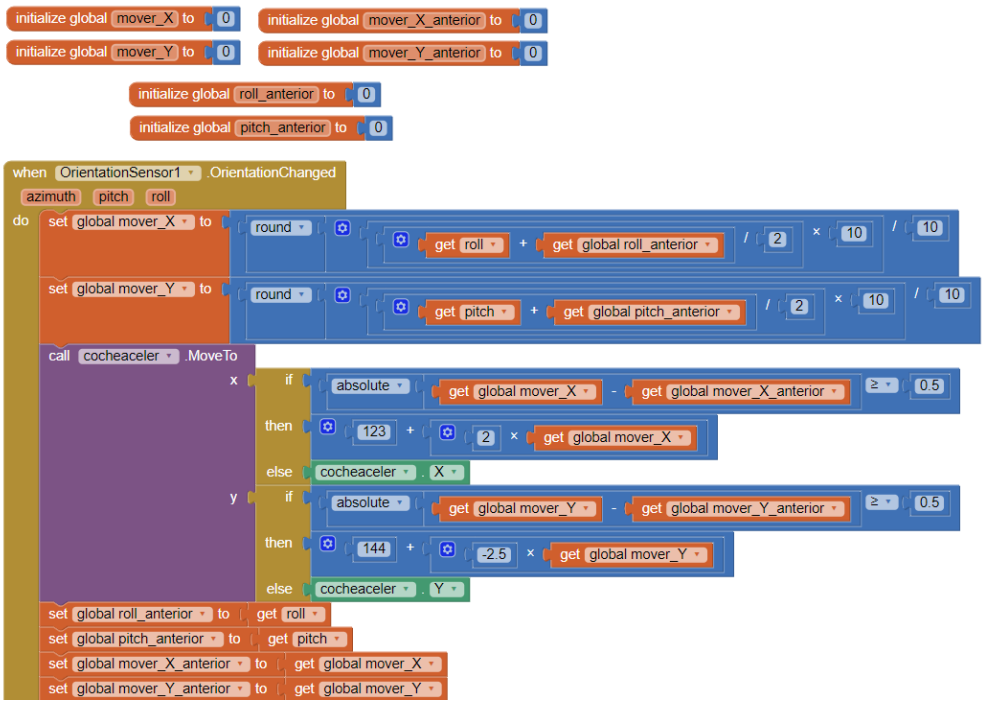
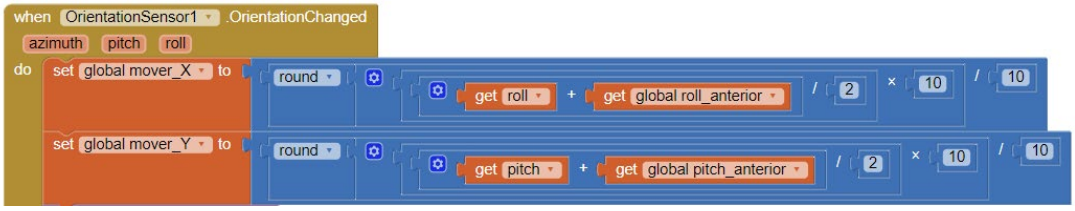


Figura 62: Programación por Bloques del movimiento del sprite del coche del Modo Giroscopio. Fuente: Propia

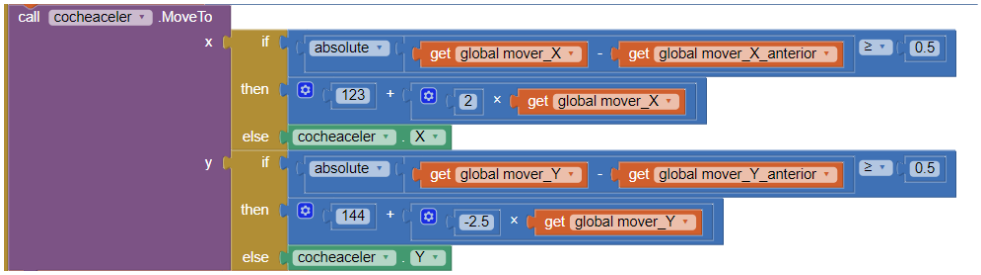
Los diferentes procesos que sigue esta parte del programa se listan a continuación:

- Se inicializan las variables globales que se utilizan en el programa.

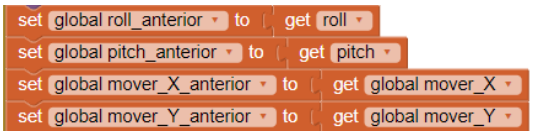
- Cuando hay un cambio en el sensor de orientación (Giroscopio) del teléfono *smartphone* (“*when Orientation Sensor. Orientation Changed*”), se realiza el cálculo de la media entre el valor anterior y el actual redondeado a un 1 decimal de los ángulos Roll y Pitch.



- Se compara la diferencia entre el valor a mover y el anterior, en caso de ser mayor o igual que un threshold, se aplica el movimiento al *sprite* con una sensibilidad de 2 en el eje X (Roll) y una sensibilidad de -2.5 en el eje Y (Pitch). Además, la posición central del canvas está referenciada por la posición X,Y [123,144] de éste.



- Finalmente, una vez aplicado el movimiento, se actualizan las variables de los estados anteriores para próxima ejecución.



- <u>Animación de los *sprites* de las flechas del canvas:</u> Una vez establecido el control del movimiento del *sprite* de la imagen del vehículo, se ha de programar las colisiones entre los *sprites*. De esta forma, cuando la imagen del vehículo se sitúe encima de una flecha, por la modificación de los ángulos Roll y Pitch, ésta se ilumine indicando la trayectoria que adquirirá el vehículo de 3 ruedas.

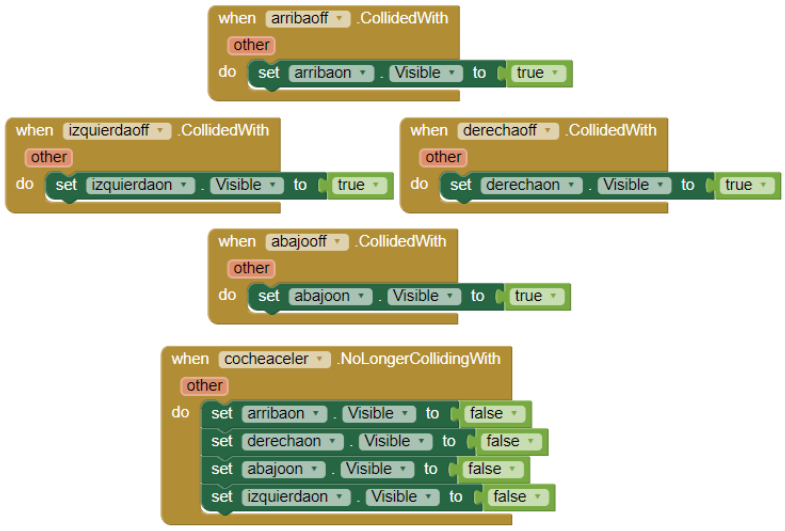
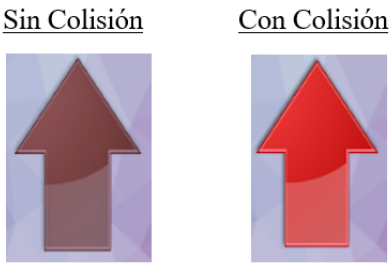


Figura 63: Programación por Bloques de los sprites de las flechas del Modo Giroscopio. Fuente: Propia

En Figura 63 se puede observar que cuando se produce una colisión entre el *sprite* de la imagen del vehículo y algún *sprite* de una flecha, se hace visible la flecha iluminada. En caso de haber colisiones, las flechas iluminadas desaparecen.



8.4.2.3. Cálculo y envío de datos Roll y Pitch

Esta parte realiza el cálculo de los datos de Roll y Pitch con 2 decimales, mediante los valores de entrada del módulo giroscopio tipo MEMS, y posteriormente el envío de éstos al PIC del vehículo cada 250ms (Figura 64).

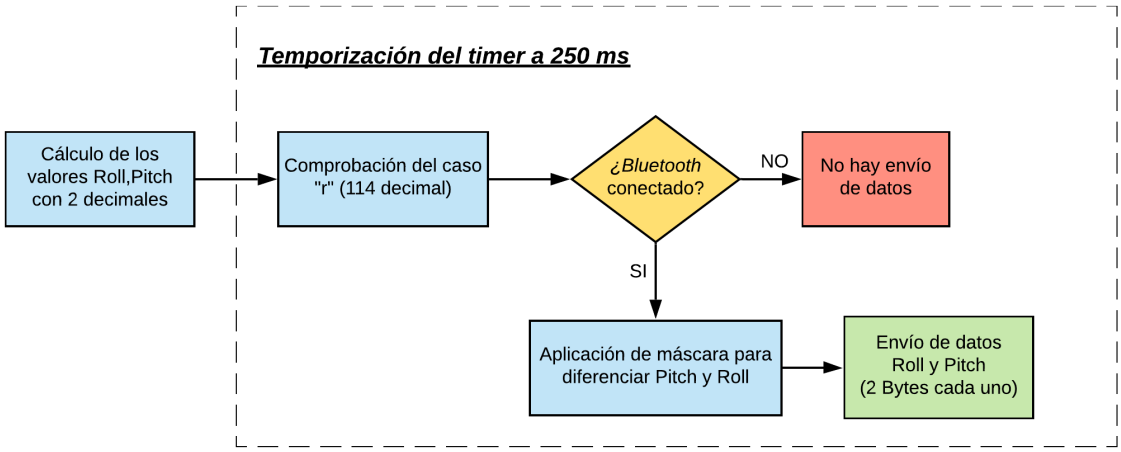


Figura 64: Diagrama de Bloques del envío de datos al PIC en el Modo Giroscopio. Fuente: Propia

- Cálculo de los valores Pitch y Roll con 2 decimales: Antes de explicar la programación por bloques de esta parte, es importante saber los rangos de valores de los ángulos Roll y Pitch obtenidos por el giroscopio y su precisión.

Figura 65: Ejemplo valores Roll y Pitch obtenidos por el giroscopio tipo MEMS del teléfono. Fuente: Propia

En la Figura 65, se observa que la precisión del giroscopio tipo MEMS del dispositivo *smartphone* es de 5 decimales y los rangos de valores de los ángulos realizando diferentes pruebas son los siguientes:

Roll → [-90.00000, 90.00000]

Pitch → [-90.00000, 90.00000]

Una vez conocida la precisión y rango de valores de los dos ángulos, se realiza la programación correspondiente para el cálculo y envío de éstos.

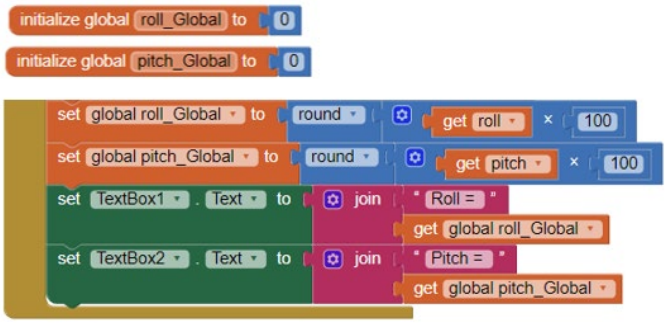


Figura 66: Programación por Bloques de la parte de cálculo del Roll y Pitch del Modo Giroscopio. Fuente: Propia

Los bloques de instrucciones de la Figura 66, se ejecutan cuando el sensor giroscopio cambia su orientación ("*when Orientation Sensor. Orientation Changed*"). En primer lugar, se inicializan las variables a utilizar, una vez inicializadas se realiza la multiplicación por 100 y el posterior redondeo con el objetivo de obtener valores enteros de los ángulos Pitch y Roll con 2 decimales. El resultado del rango de valores se puede observar en la siguiente figura de manera más gráfica (Figura 67).

Finalmente, también se muestras los valores enteros calculados del Roll y Pitch en 2 cuadros de textos diferentes ("TextBox1" y "TextBox2").

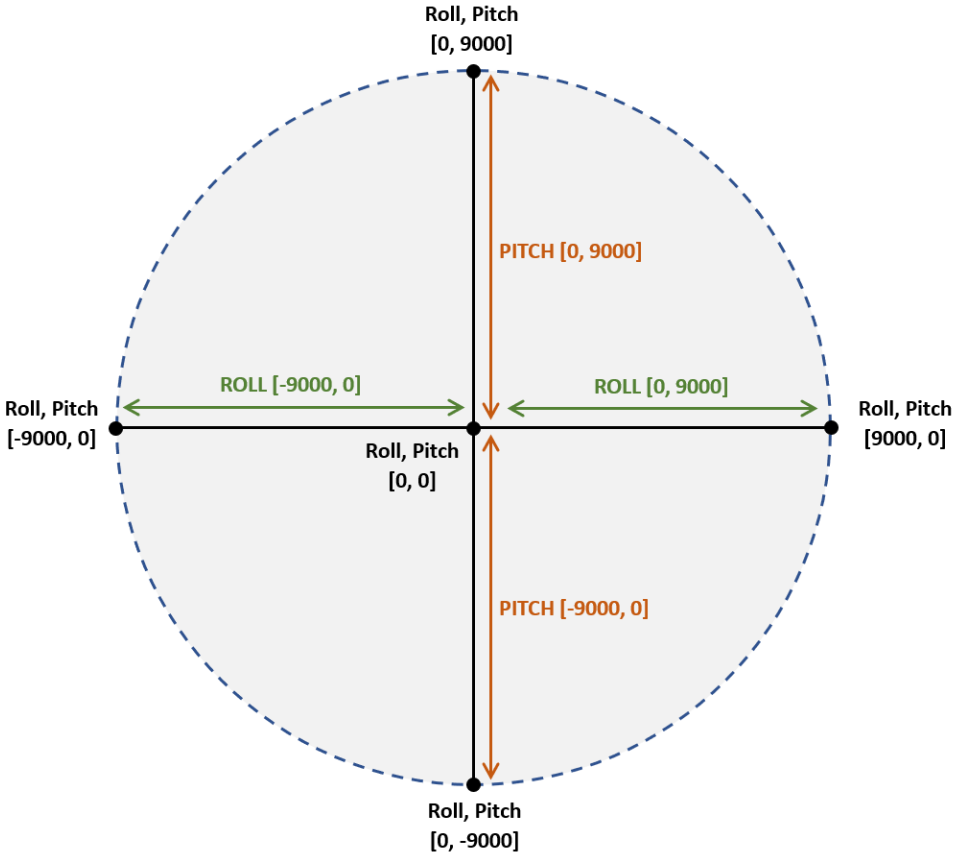


Figura 67: Rango de valores obtenidos por el giroscopio de los ángulos Roll y Pitch con 2 decimales. Fuente: Propia

Como se observa en la Figura 67, el rango de valores de los datos Roll y Pitch obtenidos por el giroscopio ha pasado a ser el siguiente:

Roll → [-9000, 9000]

Pitch → [-9000, 9000]

De esta forma se trabajará con números enteros evitando los datos tipo "float" y con una precisión de 2 decimales suficiente para el control del vehículo. Además, hay que tener en cuenta que las variables tendrán una dimensión de 16bits mínimo para poder contener estos rangos y serán del tipo "*signed*", es decir, que se tendrá en cuenta el signo. La tabla de los posibles tipos de variables a usar se muestra a continuación (Figura 68).

Type	Size	Minimum	Maximum
int	16 bits	-32,768	32,767
short	16 bits	-32,768	32,767
short long	24 bits	-8,388,608	8,388,607
long	32 bits	-2,147,483,648	2,147,483,647

Figura 68: Tipos de variables posibles para la implementación del programa. Fuente: Propia

- <u>Envío de los datos Pitch y Roll con 2 decimales:</u> En este punto de programación hay que tener en cuenta diferentes aspectos listados a continuación:

    - Los datos enviados se leerán desde el programa C, con una variable común, por lo que habrá que diferenciar el Roll y Pitch.

    - En caso de querer volver a la Pantalla de Menú Principal se enviará un valor que tendrá que ser excluido de los posibles valores obtenidos por el giroscopio y de esta manera evitar fallos.

    - Se realizará un envío eficiente de los datos, es decir, se enviará el mínimo número de bytes posible.

Teniendo en cuenta los objetivos anteriores, se aplica una codificación de los datos óptima para una variable de 16bits tipo "*signed int o int*", tal que, ambos datos Roll y Pitch son diferenciados mediante la aplicación de una máscara que utiliza los 2 bits más significativos de la variable (Bits 15 y 14 de la Figura 65).

	MÁSCARA																	
Variable	D_{15}	D_{14}	D_{13}	D_{12}	D_{11}	D_{10}	D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	*HEX*	*Decimal*
ROLL (+)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x0000	0
	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0x3FFF	16383
PITCH (+)	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x4000	16384
	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0x7FFF	32767
PITCH (-)	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x8000	-32768
	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0xBFFF	-16385
ROLL (-)	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0xC000	-16384
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0xFFFF	-1

Figura 69: Codificación con máscara de los datos Roll y Pitch a enviar en Modo Giroscopio. Fuente: Propia

Respecto a la Figura 69, para aplicar la máscara, a la hora de enviar los datos del ángulo Pitch se ha de sumar o restar el valor 16384 (*0100 0000 0000 0000* en binario) cuando sus valores sean positivos o negativos respectivamente.

<u>Ejemplo:</u>

Pitch = **50** → Pitch = 16384 + 50 = **16434**

Pitch = **-50** → Pitch = -16384 – 50 = **-16434**

Una vez explicada la codificación de los datos que se va a seguir para enviar los ángulos Pitch y Roll con tipo entero con 2 decimales, se realiza la programación por bloques de esta parte (Figura 70).

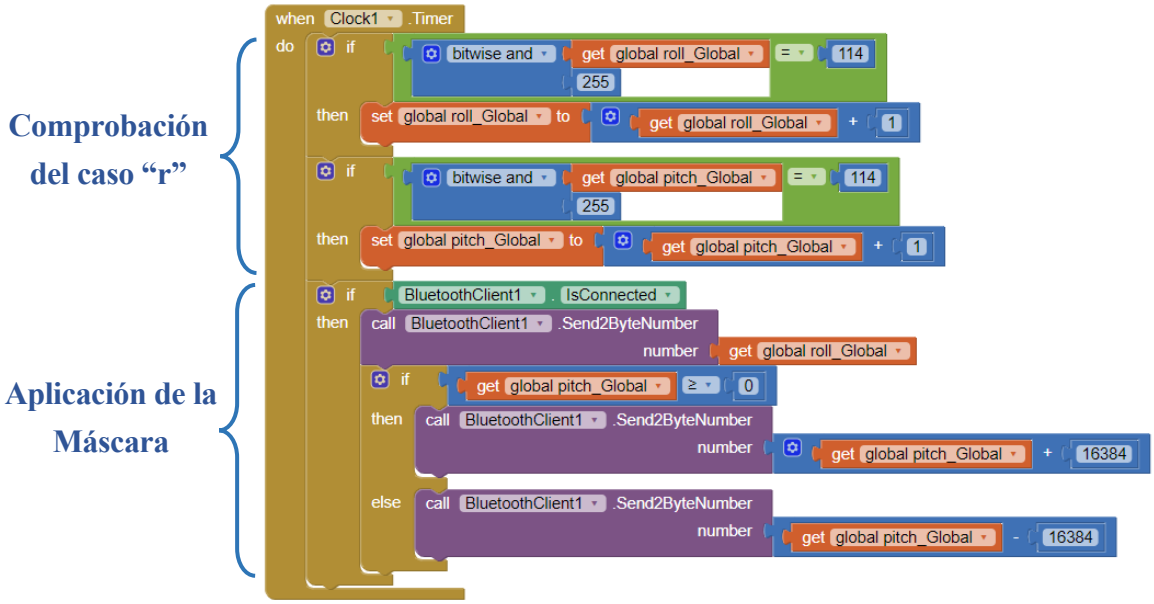


Figura 70: Programación por Bloques del envío de los datos Roll y Pitch del Modo Giroscopio. Fuente: Propia

Una vez el temporizador ("Clock1.Timer") ha terminado la cuenta de 250ms, se realiza en primer lugar la comprobación de que el byte menos significativo de los datos Roll y Pitch no coincide con el carácter *ASCII* de la letra "r" (114 en decimal) dado que este valor es el utilizado para volver a la Pantalla de Menú Principal, en caso de coincidir, se incrementa el valor en 1 (esta modificación no es significativa).

Finalmente, tras haber comprobado el caso "r", si la conexión vía *Bluetooth* está activa, se realiza el envío de datos aplicando la máscara a los datos Roll y Pitch. Los datos del Roll no se modifican, en cambio se suma/resta el valor 16384 a los datos del Pitch comprobando previamente si son positivos/negativos.

8.4.2.4. Retroceso a la Pantalla de Menú Principal

Como se ha comentado anteriormente, el valor que se ha utilizado para el retorno de la Pantalla del Modo Giroscopio al Menú Principal es el carácter *ASCII* de la letra "r" (114 en decimal). Además, este mismo valor se utiliza del mismo modo en el retorno de la Pantalla de Modo Libre.

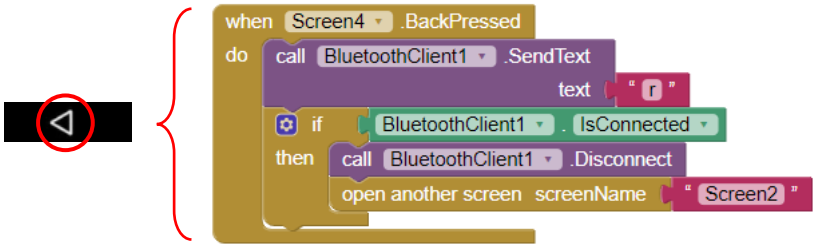


Figura 71: Programación por Bloques del retorno al Menú Principal desde el Modo Giroscopio. Fuente: Propia

La Figura 71 muestra que una vez es presionado el botón "*Back*" que incorpora el dispositivo *smartphone*, en caso de que el *Bluetooth* esté conectado correctamente con el vehículo, esta comunicación se desconectará y posteriormente se retornará a la Pantalla del Menú Principal ("*Screen2*").

8.4.3. Resultado

Tras finalizar los apartados de diseño y programación de la Pantalla del Modo Giroscopio, el resultado que se obtiene es el siguiente.

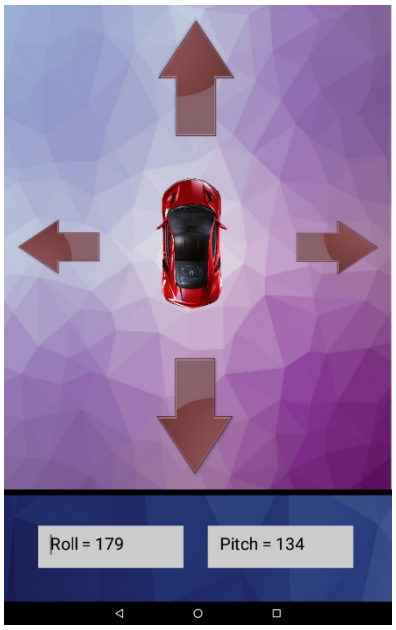


Figura 72: Resultado del Modo Giroscopio. Fuente: Propia

La Figura 72 corresponde con una posición estática del teléfono *smartphone*, en la cual el movimiento del vehículo de 3 ruedas es nulo (Roll = 1,79º, Pitch = 1,34º)

A continuación se muestran los resultados de los 4 tipos de movimientos principales del vehículo (Figura 73).

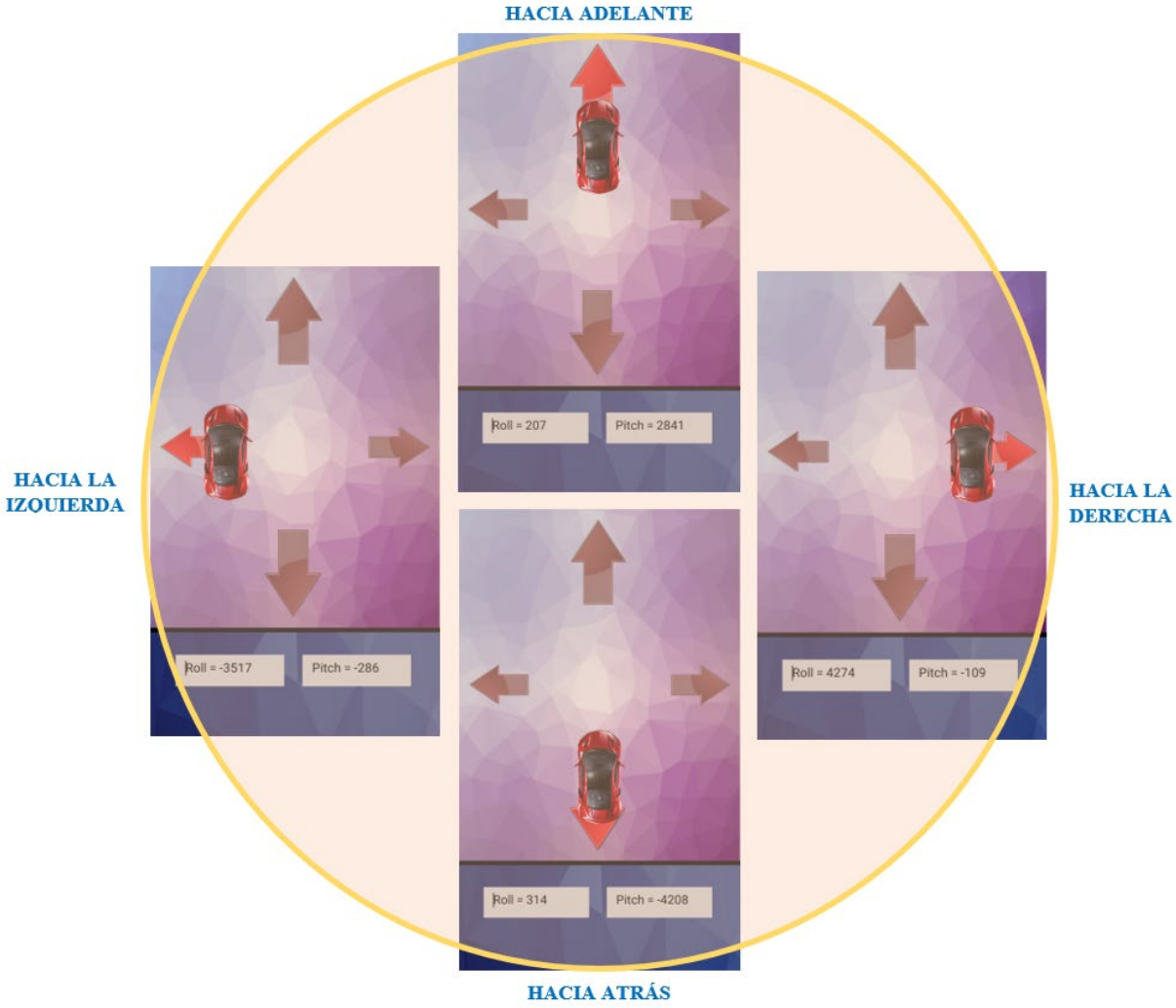


Figura 73: Resultado de los 4 tipos de movimientos principales en el Modo Giroscopio. Fuente: Propia

Respecto a los resultados observados hay que destacar que este modo posibilita la realización de una gran multitud de movimientos y de esta forma, poder regular tanto la trayectoria como la velocidad del vehículo de independiente y bastante precisa.

Además, el sencillo diseño de la interfaz gráfica mediante los *layouts*, facilita el entendimiento de los movimientos realizados por el usuario en todo momento.

8.5. Pantalla del "Modo Libre"

La Pantalla del Modo Libre permite al usuario controlar tanto la velocidad como la trayectoria del vehículo a través de su interacción con diferentes inputs y la comunicación *Bluetooth* configurada en el Menú Principal.

8.5.1. Diseño

El diseño de esta pantalla cuenta con 5 botones y una *slider-bar*. Los botones se corresponden con los movimientos: hacia adelante, hacia atrás, giro a derechas, giro a izquierdas y paro. Por otro lado, la *slider-bar* permite configurar la velocidad del coche durante las trayectorias que realiza, modificando el ciclo de trabajo en los motores de forma conjunta.

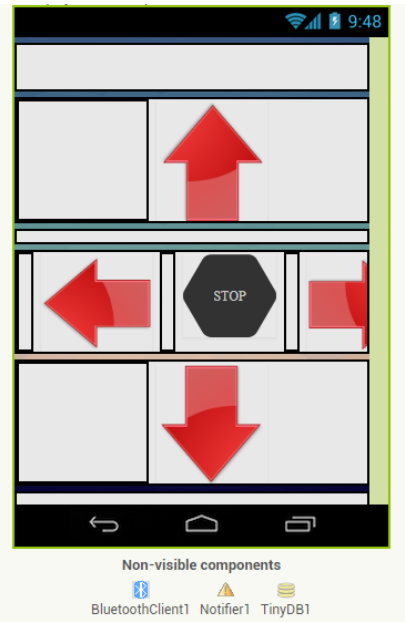


Figura 74: Diseño de la Pantalla del Modo Libre. Fuente: Propia

En la Figura 74 se observa que los módulos utilizados son los mismos que en la Pantalla de Menú Principal. Esto se debe a que sólo se utiliza la comunicación *Bluetooth* multi-pantalla para el envío de datos generados por los inputs, sin necesidad de utilizar ningún otro módulo/sensor del dispositivo *smartphone*.

Otro aspecto importante en el diseño de esta pantalla, es la utilización de *layouts* para generar una interfaz gráfica cómoda y ordenada para el usuario.

8.5.2. Programación

La programación de esta pantalla se divide en tres diferentes partes (Figura 75), la reconexión de la comunicación *Bluetooth* multi-pantalla, el retroceso a la pantalla de Menú Principal mediante el botón "*back*" y el envío de datos mediante los botones/*slider-bar* al PIC.

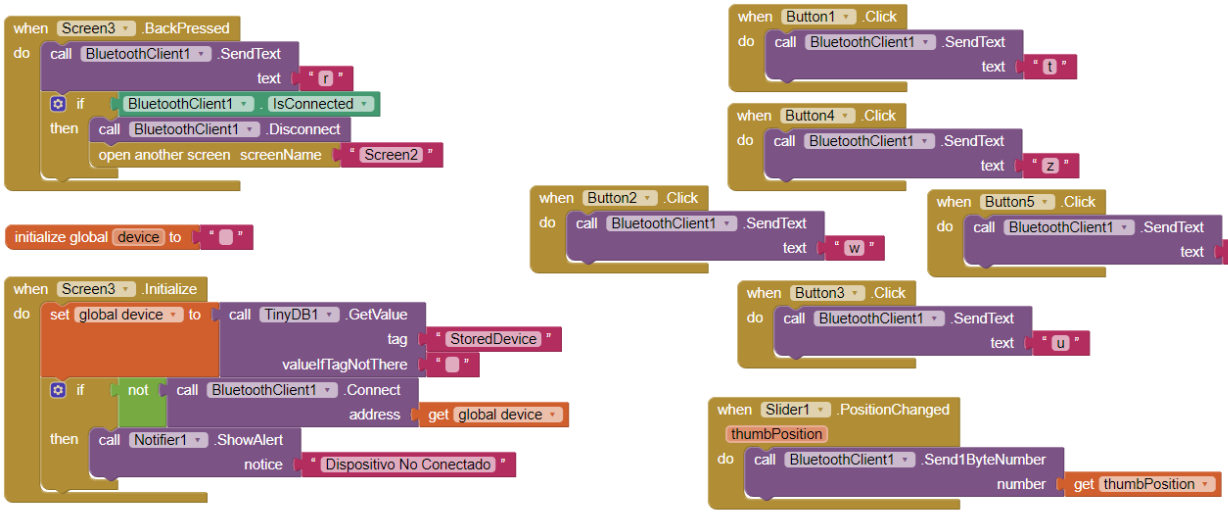


Figura 75: Programación por Bloques de la Pantalla del Modo Libre. Fuente: Propia

Tanto la parte de reconexión del *Bluetooth* como el retorno al Menú Principal, son idénticas a las que se explicaban anteriormente en la Pantalla del Modo Giroscopio en los apartados 8.4.2.1 y 8.4.2.4 respectivamente. Por lo tanto, solo se explicará el envío de datos generados por los inputs implementados en el diseño(Figura 76).

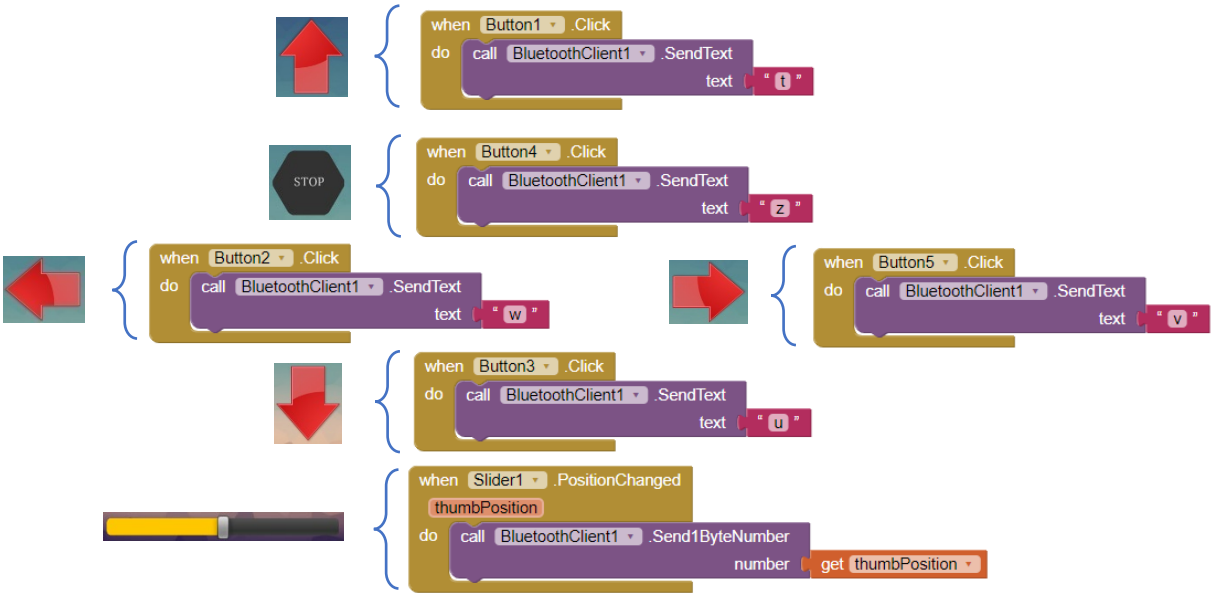


Figura 76: Programación por Bloques de los botones y slider-bar del Modo Libre. Fuente: Propia

Como se puede apreciar en la Figura 76, cada uno de los 5 botones tienen un carácter ASCII asociado:

- Flecha arriba (↑): Carácter **" t "** (116 en decimal)
- Flecha abajo (↓): Carácter **" u "** (117 en decimal)
- Flecha derecha (→): Carácter **" v "** (118 en decimal)
- Flecha izquierda (←): Carácter **" w "** (119 en decimal)
- Botón STOP: Carácter **" z "** (121 en decimal)

Los valores en decimal asociados a los diferentes caracteres han de ser mayores que 100, dado que la *slider-bar* se configura para generar un valor comprendido entre 0-100 (Figura 77).

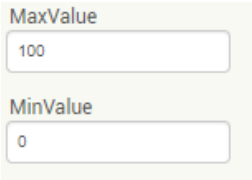


Figura 77: Configuración del rango de valores de la slider-bar del Modo Libre. Fuente: Propia

Otro aspecto importante es el valor decimal del carácter "r" utilizado en el retorno a la Pantalla del Menú Principal el cual al ser 114 tampoco influye. Por lo tanto, todos los inputs que podrá generar el usuario enviarán un único byte de datos de tipo *unsigned char* (Figura 78).

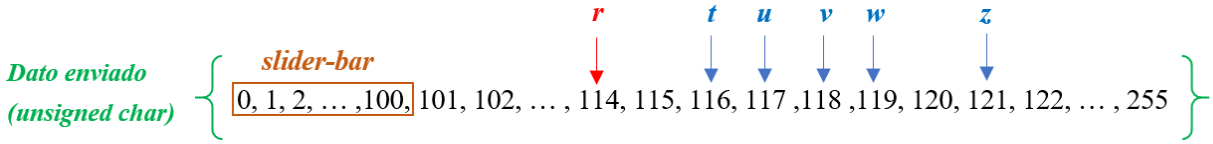


Figura 78: Byte de Datos enviado en función de los inputs generados en el Modo Libre. Fuente: Propia

8.5.3. Resultado

El resultado obtenido tras los apartados de diseño y programación de la pantalla del Modo Libre, se muestra a continuación.

Una vez iniciada la pantalla, se muestran todos los inputs con los cual el usuario puede realizar el control del vehículo desde el teléfono *smartphone* (Figura 79).

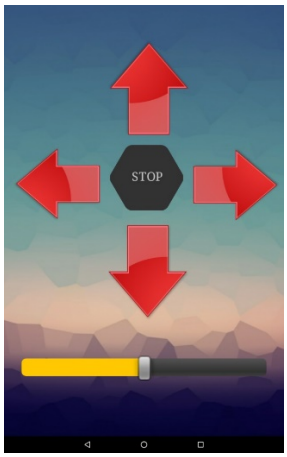


Figura 79: Resultado de la pantalla de Modo Libre. Fuente: Propia

Existen dos tipos de interacciones, la primera tiene como objetivo modificar la trayectoria del vehículo (*inputs* de botones de flechas) y la segunda interacción realiza el control de la velocidad de este (*slider-bar* y botón "STOP").

- Inputs de control de velocidad: Los inputs que se encargan de controlar la velocidad del vehículo, regulando la potencia que suministrarán los motores DC a las ruedas, son la *slider-bar* y el botón "STOP".

Figura 80: Interacción con la slider-bar en el Modo Libre. Fuente: Propia

En función de la posición escogida por el usuario de la *slider-bar* el vehículo irá más lento o más deprisa (Figura 82).

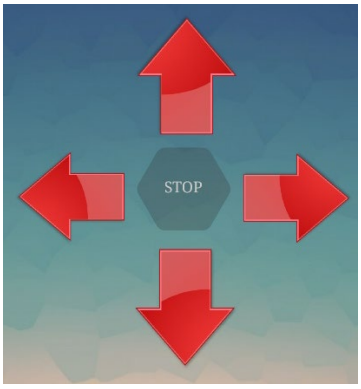


Figura 81: Interacción con el botón STOP del Modo Libre. Fuente: Propia

Accionando el botón STOP, el vehículo se detiene de manera automática.

- Inputs de trayectoria: Los 4 botones con forma de flecha tienen el objetivo de modificar la trayectoria que seguirá el vehículo de tres ruedas. El resultado al accionarlos es el siguiente (Figura 82).

Figura 82: Resultado de la trayectoria del vehículo al pulsar las diferentes flechas en el Modo Libre. Fuente: Propia

9. Código fuente C del programa del PIC

Para realizar un buen control del vehículo de tres ruedas, el microcontrolador ha de interpretar los datos recibidos a través de sus pines externos y realizar las acciones correspondientes. Para ello, se realiza la escritura del código fuente en lenguaje C del programa que implementará el PIC16F723A.

El programa internamente cuenta con 2 funciones principales (*SETUP* y *LOOP*) llamadas desde el "*main program"* y 1 función de atención a interrupciones (*Interrupt RAI*) desde donde se llama a otras 3 funciones secundarias para los cálculos necesarios en los modos "*Giroscopio*" y "*Libre*".

Las llamadas a las diferentes funciones se pueden visualizar en el siguiente diagrama de bloques (Figura 83).

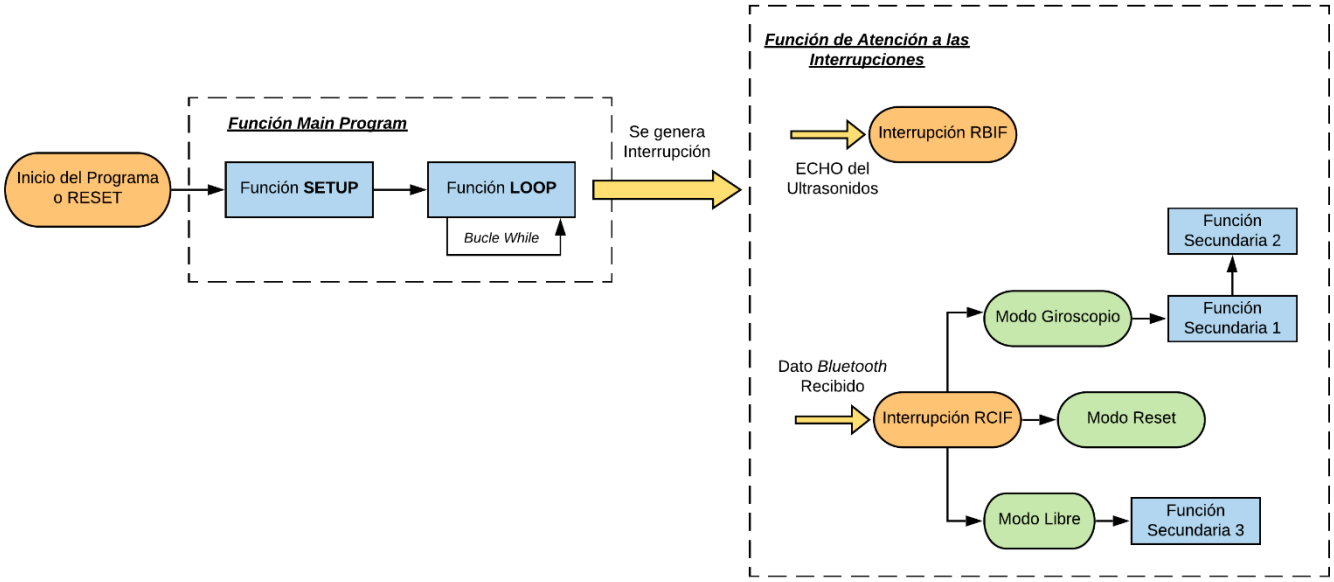


Figura 83: Diagrama de Bloques de las llamadas a las funciones del programa C. Fuente: Propia

Tal y como se observa en la Figura 83, el programa internamente cuenta con 3 modos diferentes, cada uno de ellos ejecuta una serie de instrucciones asociadas a distintas funciones, estos modos se listan a continuación.

- Modo Giroscopio: Está asociado a la "*Pantalla del Modo Giroscopio*" de la aplicación móvil para *smartphones*. Realiza la interpretación de los datos Roll/Pitch recibidos y posteriormente genera las señales PWM correspondientes en los motores DC

- Modo Libre: Está asociado a la "*Pantalla del Modo Libre*" de la aplicación móvil para *smartphones*. Realiza la interpretación de los datos enviados por los inputs de los botones/*slider-bar* y determina la velocidad/trayectoria que seguirá el vehículo.

- Modo Reset: Está asociado a la "*Pantalla de Menú Principal*" de la aplicación móvil para *smartphones*. En este modo el vehículo se encuentra parado completamente, dado que el ciclo de trabajo de los motores (*Duty Cicle*) permanece a 0%.

Una vez observado el diagrama de bloques que sigue el programa respecto a las funciones que incorpora y explicados los diferentes modos que existen dentro de este, se detallarán más en profundidad las funciones "*main program"* e "*Interrupt RAI*".

9.1. Función Main Program

La función "*main program*" sirve como punto de partida para la ejecución del programa y ha de estar presente en todos los programas escritos en lenguaje C (sin excepciones).

Este tipo de función no se declara, simplemente se define. Además, controla la ejecución del programa dirigiendo las llamadas a otras funciones de éste, las cuales han de ser definidas o al menos declaradas previamente.

```
void main(void){

    setup();                //Llamada a la función de configuración de registros

    loop();                 //Llamada a la función bucle, LOOP


}
```

Figura 84: Código fuente C de la función "main program". Fuente: Propia

En el código fuente de este programa (Figura 84), dentro de la función "*main*" se llama a las 2 funciones principales *SETUP* y *LOOP*, las cuales tienen como objetivo configurar los registros que se utilizarán en el programa (*SETUP*) y la realización de un bucle infinito de instrucciones que se ejecutará de forma continuada (*LOOP*).

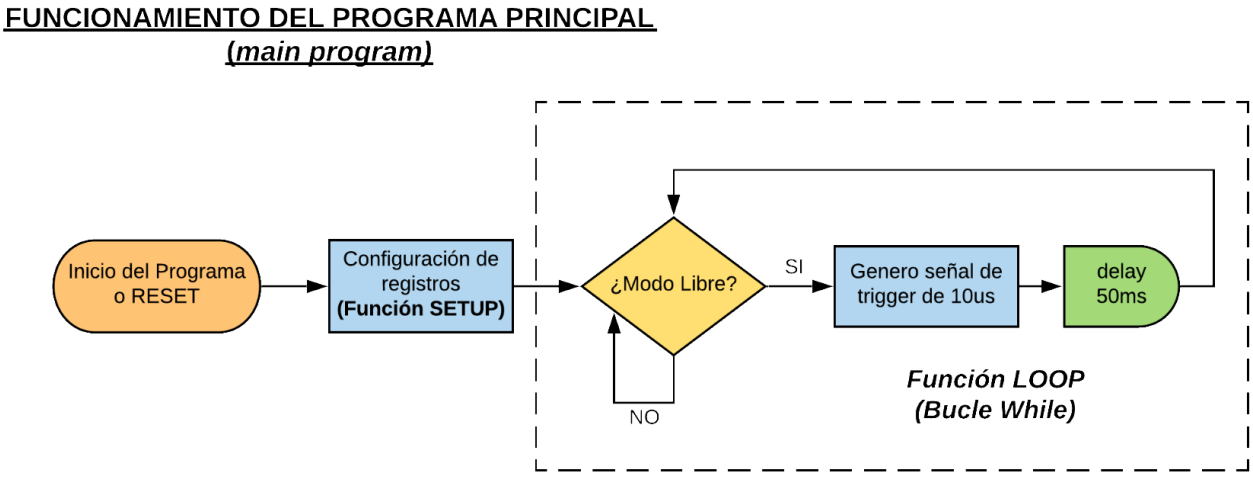


Figura 85: Diagrama de Bloques de la función "main program". Fuente: Propia

Observando la Figura 85, una vez realizado la configuración de los registros, el programa entra en bucle infinito ("*Bucle While"*). Este bucle únicamente realiza el disparo del trigger del sensor ultrasonidos siempre y cuando el programa se encuentre en "Modo Libre".

9.1.1. Función SETUP

La función *SETUP* configura uno a uno los registros internos del PIC que se utilizan durante la ejecución del programa. El código fuente se muestra a continuación (Figura 86).

```
void setup(void){       //*******FUNCIÓN SETUP********

//OSCILADOR
OSCCON = 0x30; //Frecuencia a 16MHz

//MODULO UART
UART_Init(); // Se configura el módulo UART para trabajar a
             // a 9600 baudios

//INTERRUPCIONES
GIE=1; // Enable Interrupciones Globales
PEIE=1; // Enable Interrupciones Periféricos
RCIE=1; // Enable Interrupción Dato Recibido BT (RX)
TXIE=0; // Disable Interrupción Dato Enviado (TX)
RBIE=1; // Enable Interrupción PORT B cambio de flanco (ULTRASONIDOS)

ANSELA=0b00000000; //en el 16F723A la configuracion (entradas A 0 D) es con el registro ANSELA
ANSELB=0b00000000;

//MÓDULO BT
TRISCbits.TRISC7 = 1; // RX como INPUT
TRISCbits.TRISC6 = 0; // TX como OUTPUT

//PWM
APFCONbits.CCP2SEL = 0; // CCP2 en RC1
TRISC = 0b11110000;     // RC3:RC0 Outputs (MOTORES)
TRISCbits.TRISC4 = 0; // RC4 como OUTPUT

CCP1CON=0b00001100;     //MODO PWM y resolución del ciclo de trabajo de 8bits
CCP2CON=0b00001100;     //MODO PWM y resolución del ciclo de trabajo de 8bits
T2CON=0b00000011;       //Postdivisor nos da igual, Pre-scaler=16, TMR2ON=0
CCPR1L=0;               //Ton a 0
CCPR2L=0;               //Ton a 0

PR2 = 249;      //frecuencia de motores a 1000HZ

//Entradas y Salidas
TRISCbits.TRISC4 = 0; // RC4 como OUTPUT

TMR2ON = 1; // Se activa el Timer 2

// ULTRASONIDOS
TRISAbits.TRISA0 = 0; // RA0 como OUTPUT (TRIGGER)
TRISBbits.TRISB5 = 1; // RB5 como INPUT (ECHO)
IOCBbits.IOCB5 = 1; // Se activa las Interrupciones por
                    // cambio de flanco del PORT B
T1CON=0b00110000;   // Pre-scaler=8, TMR1ON = 0

TMR1ON = 0; // Se mantiene parado el TIMER 1

}
```

Figura 86: Código fuente C de la función SETUP. Fuente: Propia

9.1.2. Función LOOP

La función *LOOP* se caracteriza por implementar un bucle infinito ("*while(1)*") el cual permite que el programa funcione de forma continuada. Cuando se produce una interrupción, la ejecución del bucle se detiene hasta que es atendida por la función "*Interrupt RAI*".

```
void loop(void){        //********FUNCIÓN LOOP********

    while(1) {  // Bucle INFINITO

        if(MODO=='M'){      // En Modo Manual se envía
            trigger = 1;    // la señal de trigger del
            __delay_us(10); // sensor Ultrasonidos
            trigger = 0;
            __delay_ms(50); // Delay para dejar tiempo
        }                   // a que se reciba el "echo"
    }
}
```

Figura 87: Código fuente C de la función LOOP. Fuente: Propia

9.2. Función Interrupt RAI

La función "*Interrupt RAI*" es programada por el usuario y tiene como objetivo atender las interrupciones generadas por los diferentes eventos del programa.

En el control del vehículo se utilizan únicamente dos interrupciones.

- Interrupción por Recepción de Datos *Bluetooth* (RCIF): Durante la comunicación vía *Bluetooth* entre el vehículo y el dispositivo *smartphone*, el usuario envía datos que son recibidos por el "Módulo HC-05" (7.2). Posteriormente estos datos llegan al pin RX del PIC16F723 el cual genera una interrupción cuando la recepción del byte de datos ha sido correcta (RCIF = 1).

- Interrupción por "*ECHO"* del sensor de ultrasonidos (RBIF): Para una mayor seguridad en el "Modo Libre", el vehículo incorpora un sensor de ultrasonidos capaz de medir distancias con una precisión suficiente para las necesidades del proyecto. De manera que, cuando el pulso de *trigger* emitido se recibe de nuevo tras rebotar en un objeto (*ECHO =1)* o deja de recibirse (*ECHO =0*), se genera una interrupción (RBIF = 1).

La implementación de estas funciones y sus diagramas de funcionamiento se explican en los apartados siguientes (9.2.1 y 9.2.1.4).

9.2.1. Interrupción por Recepción de Datos Bluetooth (RCIF)

Es la interrupción más importante de todas, dado que permite realizar las acciones de control en el vehículo, a partir de las decisiones que toma el usuario desde el dispositivo *smartphone*.

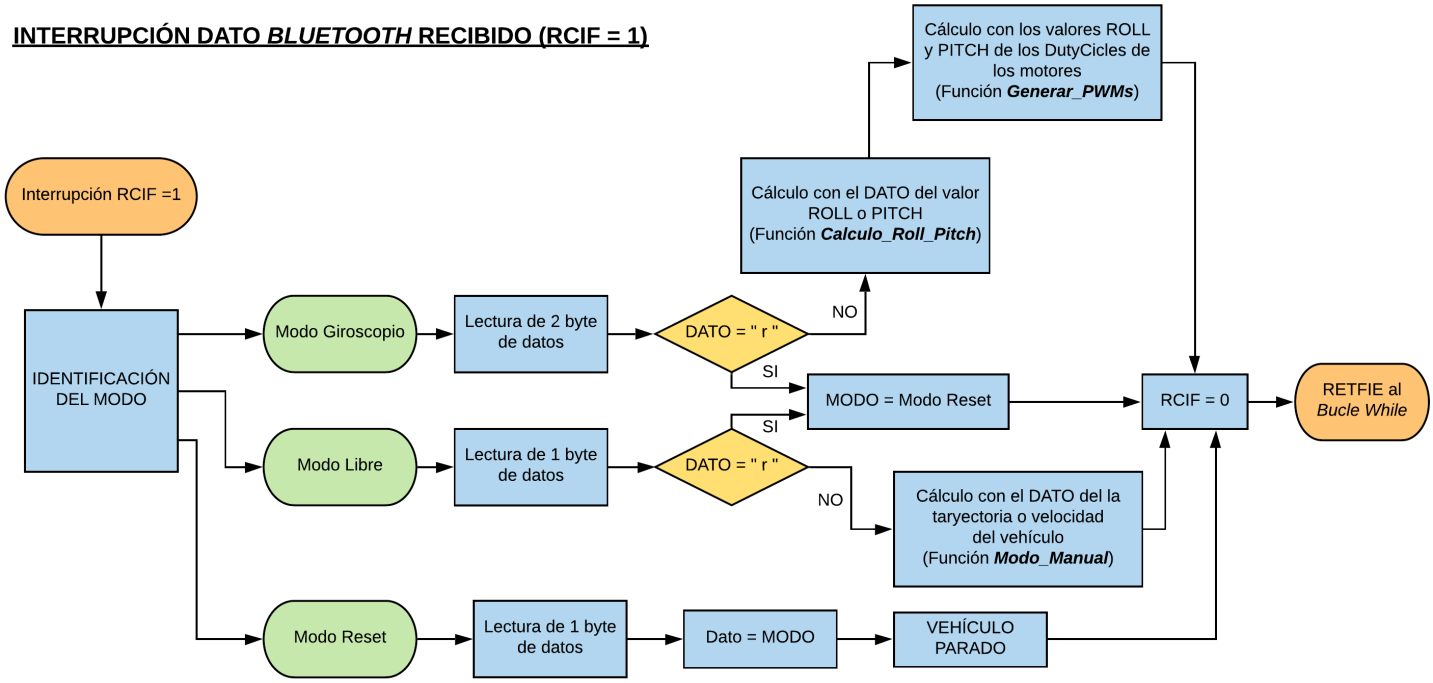


Figura 88: Diagrama de Bloques de la subrutina de interrupción RCIF. Fuente: Propia

En primer lugar, con respecto a la Figura 88, se realiza la lectura de los datos dependiendo el modo en el que se encuentre el programa. La lectura en los modos "Libre" y "Reset" es de solo un byte, mientras que en el "Modo Giroscopio" es de dos debido al tamaño de los datos de los ángulos Roll y Pitch.

Además, para la realización de la lectura de los datos previamente se ha tenido que configurar de manera correcta el módulo *AUSART* del microcontrolador. Para ello, se utiliza el fichero de cabecera (del inglés *header file*) "***uart.h***" que incluye diferentes funciones para la configuración, recepción y transmisión de datos de este módulo.

Por otro lado, una vez realizada la lectura de los datos en los modos "Libre" y "Giroscopio", son llamadas las funciones secundarias correspondientes que realizan el cálculo y control de las señales que alimentan los motores DC de cada modo.

El código fuente implementado para la interrupción RCIF es el siguiente (Figura 89)

```
if (PIR1bits.RCIF){ // INTERRUPCIÓN DATO RECIBIDO (RX)

            switch(MODO){ // Este Switch distigue el Modo en el que se encuentra
                          // el vehículo y lee los datos del buffer RCREG ("uart.h")

                case 'G':   //***** GIROSCOPIO ******
                    valor = UART_Read_2byte(); // Lectura de 2 Bytes consecutivos
                    if(valor == 'r') MODO = 'r'; // Caso RESET (Botón BACK)
                break;

                case 'M':   //***** MANUAL ******
                    valor = UART_Read(); // Lectura de 1 Byte
                    if(valor == 'r') MODO = 'r'; // Caso RESET (Botón BACK)
                break;
                            //***** RESET (Pantalla Menú Principal) ******
                case 'r':   // 114 en ASCII
                    MODO = UART_Read(); // Lectura de 1 bytes (Modo)
                break;
            }

            switch(MODO){ // Una vez leidos los datos en función del
                          // Modo, se aplica las funciones correspondientes

                case 'G': // LLAMADA A FUNCIONES DEL MODO GIROSCOPIO
                    Calculo_Roll_Pitch(valor);
                    Generar_PWMs(ROLL,PITCH);
                break;

                case 'M': // LLAMADA A FUNCIONES DEL MODO LIBRE
                    Modo_Manual(valor);
                break;

                case 'r': // CASO RESET (MENU PRINCIPAL, TODO PARADO)
                RC0=0; RC3=0; CCPR1L=0;CCPR2L=0;
                break;
            }

PIR1bits.RCIF = 0;  //Desactivamos la bandera de recepción en el buffer de entrada del USART

}
```

Figura 89: Código fuente C de la Interrupción RCIF. Fuente: Propia

A continuación se explicarán las tres funciones secundarias que se aplican en esta parte del programa (2 del "Modo Giroscopio" y 1 del "Modo Libre") y el fichero de cabecera "*uart.h*" utilizado para las acciones de la comunicación vía *Bluetooth*.

9.2.1.1. Fichero de cabecera "uart.h"

Los ficheros de cabecera (*header* files), son archivos incluidos de forma automática por el compilador al procesar algún otro archivo en código fuente. Además, puede contener subrutinas, variables u otros identificadores.

El archivo "*uart.h*" utilizado en el programa de control del vehículo, consta de 3 subrutinas diferentes que serán llamadas durante la ejecución de este.

- Subrutina UART_Init(): Realiza la configuración del módulo AUSART interno del PIC16F723A, estableciendo la velocidad a 9600 baudios y habilitando la recepción/transmisión de datos de forma asíncrona.

```
UART_Init(){

    TXSTA=0x20; // BRGH = 0
    SPBRG=25;   // Fosc = 16MHz, baud 9600 y BRGH = 0(MANUAL)
    RCSTA=0x90;
    //Baud Rate = Fosc/(64*(SPBRG+1))

  /*** Bits configurados de los registros TXSTA y RCSTA *******

    SYNC = 0;      // Selección del Modo Asíncrono del AUSART
    SPEN = 1;      // AUSART ON
    CREN = 1;      // Habilita la Recepción Continua
    TXEN = 1;      // Transmisión Habilitada
*/
}
```

Figura 90: Código fuente C de la subrutina UART_Init() de fichero "uart.h". Fuente: Propia

- Subrutina UART_Read(): Realiza la lectura de un byte de datos, el cuál una vez es recibido se almacenada en el buffer RCREG y provoca un flanco de subida en RCIF.

```
char UART_Read()
{
  while(!RCIF); //Mientras RCIF sea 0 lo hace
  return RCREG;
}
```

Figura 91: Código fuente C de la subrutina UART_Read() del fichero "uart.h". Fuente: Propia

❖ Subrutina UART_Read_2byte(): Realiza la lectura de 2 bytes de datos consecutivos de datos. Su funcionamiento es similar al de la subrutina UART_Read() pero en este caso se realiza también la comprobación del caso "r" antes de leer el segundo byte de datos más significativo.

```
int UART_Read_2byte()
{
  int Output = 0;

  while(!RCIF);       //Mientras RCIF sea 0 lo hace
  Output = RCREG;

  if (Output != 'r'){

    RCIF = 0;             // Se pone a 0 para esperar el
                        // siguiente Byte de datos
    while (!RCIF);
    Output+= RCREG<<8; // Se pone el siguiente Byte más significativo

  }

  return Output;
}
```

Figura 92: Código fuente C de la subrutina UART_Read_2byte() del fichero "uart.h". Fuente: Propia

En el código fuente de la Figura 92, se observa en primer lugar se recibe el byte menos significativo del dato a recibir (b7-b0) y posteriormente el más significativo (b15-b8), el cual se desplaza 8 posiciones hacia la izquierda para completar la variable Output de 16 bits. Una mejor explicación gráfica se muestra en la siguiente figura (Figura 93).

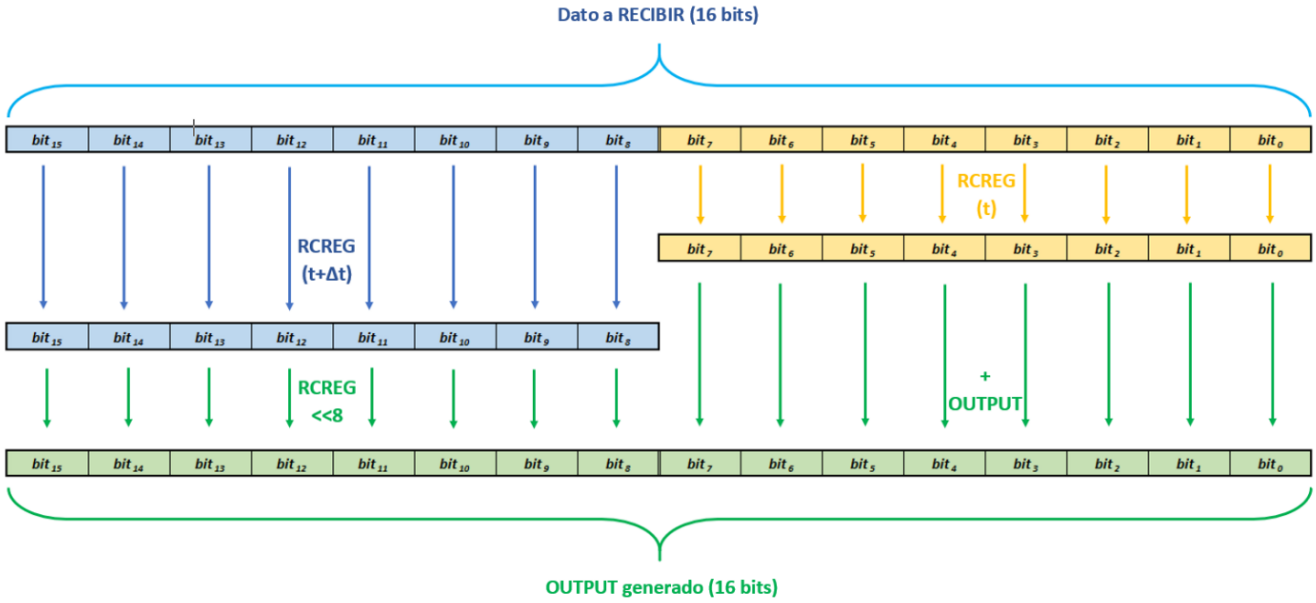


Figura 93: Lectura de 2 bytes de datos de la subrutina UART_Read_2byte() gráficamente. Fuente: Propia

9.2.1.2. Función Cálculo_Roll_Pitch

Los datos Roll y Pitch enviados con el teléfono *smartphone* desde el "Modo Giroscopio" (variable *MODO = 'G'*) son codificados mediante una máscara para ser diferenciados, explicado anteriormente en el apartado 8.4.2.3. De tal modo que se ha revertir el proceso y extraer el dato Roll o Pitch en función de la máscara identificada.

El código fuente que realiza esta asignación se muestra a continuación (Figura 94).

```
void Calculo_Roll_Pitch(int dato){

unsigned char bits_SUP;          // Se mira la máscara del dato
                                 // y en función de su valor se
        bits_SUP = dato>>14;     // obtiene si es ROLL o PITCH

        switch(bits_SUP){

            case 0:     //ROLL POSITIVO     // En los casos ROLL
            ROLL = dato;                    // el dato no se modifica
            break;

            case 255:   //ROLL NEGATIVO
            ROLL = dato;
            break;

            case 1:     //PITCH POSITIVO    // En los casos PITCH
            PITCH = dato - 16384;           // se resta/suma 16384
            break;

            case 254:   //PITCH NEGATIVO
            PITCH = dato + 16384;
            break;

        }

}
```

Figura 94: Código de la función secundaria Calculo_Roll_Pitch. Fuente: Propia

Respecto al código de la Figura 94, hay que tener en cuenta que, al desplazar 14 posiciones hacia la derecha el dato, el bit introducido por la izquierda corresponde a bit más significativo del dato (b15) y, por tanto, los valores decimales cambian (Figura 95).

	MÁSCARA				
Variable	D_{15}	D_{14}	→	Binario	Decimal
ROLL (+)	*0*	*0*	→	0000 0000 0000 0000	0
PITCH (+)	*0*	*1*	→	0000 0000 0000 0001	1
PITCH (-)	*1*	*0*	→	1111 1111 1111 1110	254
ROLL (-)	*1*	*1*	→	1111 1111 1111 1111	255

Figura 95: Asignación de variables ROLL y PITCH en función de la Máscara. Fuente: Propia

9.2.1.3. Función Generar_PWMs

Una vez identificados los valores de los ángulos Roll y Pitch, se realiza la llamada a la función "Generar_PWMs", la cual ejecuta todos los cálculos para definir la nueva velocidad y trayectoria del vehículo.

En primer lugar, se realiza el ajuste de sensibilidad en el control del "Modo Giroscopio". Para ello, se definen las diferentes constantes del compilador que establecen unos rangos en los cuales los valores de Roll y Pitch han de estar para aplicar el algoritmo de control correspondiente.

```
#define PITCH_H 6000
#define PITCH_L 500
#define ROLL_H 6000
#define ROLL_L 500
```

Figura 96: Constantes definidas para los Rangos de valores del Roll y Pitch del Modo Giroscopio. Fuente: Propia

Roll: [-6000, -500] U [500, 6000]

Pitch: [-6000, -500] U [500, 6000]

Modificando la Figura 67 con el nuevo rango de valores, se obtiene el siguiente gráfico (Figura 98).

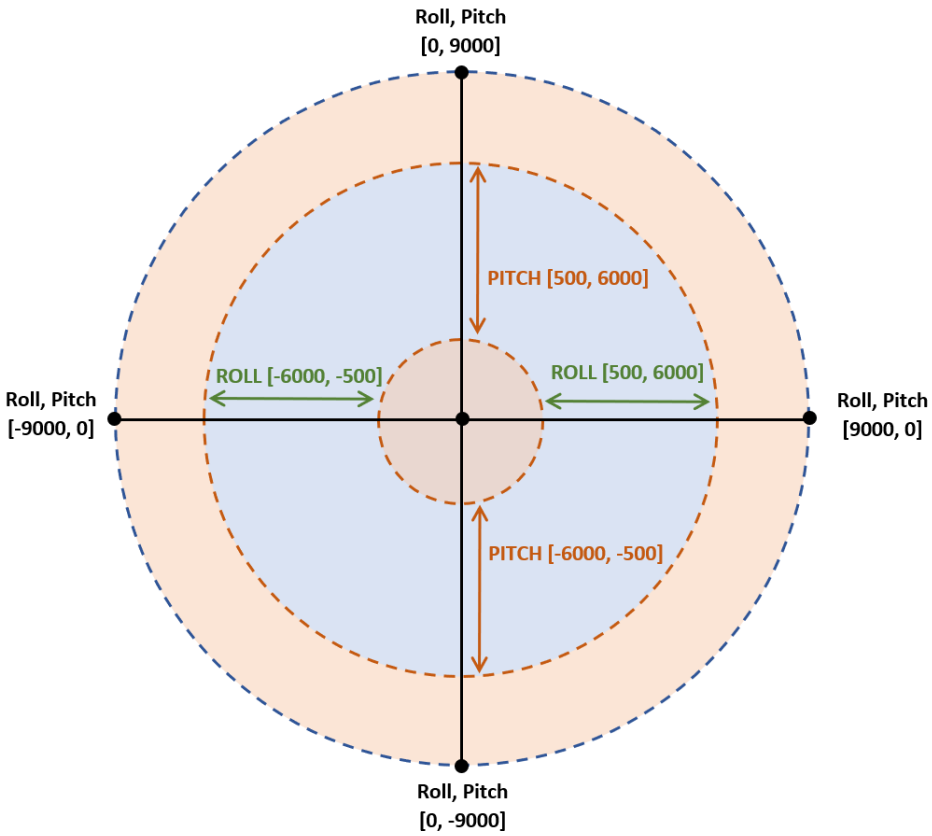


Figura 97: Rango de valores actualiza de los ángulos Roll y Pitch donde se aplicará el algoritmo de control. Fuente: Propia

Una vez realizada la definición de las constantes de los rangos del Roll y Pitch, dado que ningún sensor es en la práctica ideal, se ha calibrar el giroscopio tipo MEMS que incorpora el teléfono *smartphone*. En este proyecto, los valores obtenidos fueron los siguientes, $Offset_{ROLL} = 1{,}79º$; $Offset_{PITCH} = 1{,}79º$ (Figura 98)

```
//******** OFFSETS *****

roll = roll -179;
pitch = pitch -179;
```

Figura 98: Offsets del sensor giroscopio tipo MEMS del teléfono smartphone. Fuente: Propia

El siguiente paso es la realización del algoritmo de control, para ello hay que realizar el estudio de cómo aplicar los datos de Roll y Pitch de rangos entre [-9000, 9000] a valores del ciclo de trabajo de cada motor entre [0, 250] (tiene este rango por cómo está definido el periodo PWM a partir del Timer2).

- **Roll:** Determina el giro del vehículo, existen 2 casos diferentes:

Valores entre [-9000, 0] (inclinación hacia la izquierda del *smartphone*): El vehículo girará hacia la izquierda, para ello, en función de la inclinación el motor izquierdo perderá relevancia de forma lineal mientras el derecho se mantiene al 100%.

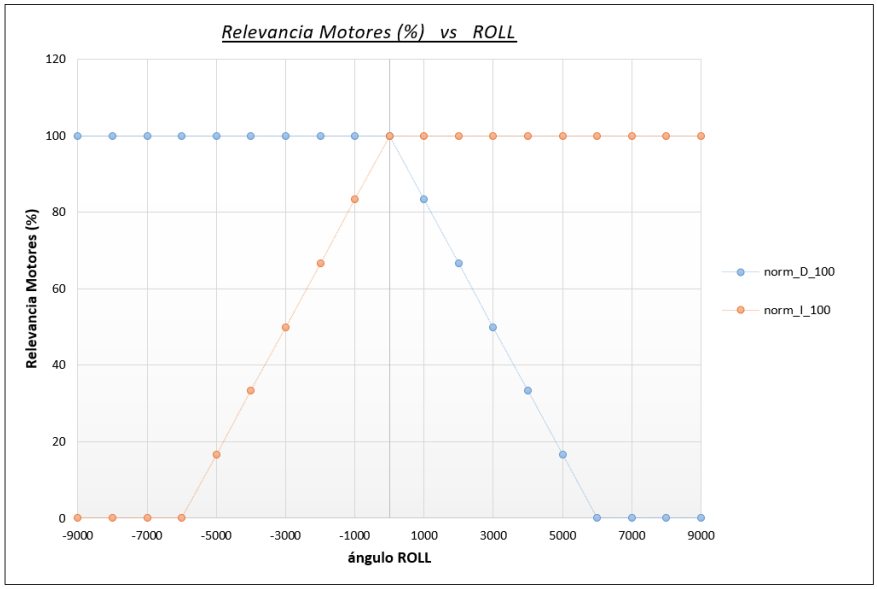
$$\begin{cases} Relevancia\ Motor\ Izquierdo(\%) = 100 & Para\ Roll \in (-500, 0] \\ Relevancia\ Motor\ Izquierdo(\%) = 100 - \left(\dfrac{Roll \cdot 100}{-6000}\right) & Para\ Roll \in [-6000, -500] \\ Relevancia\ Motor\ Izquierdo(\%) = 0 & Para\ Roll \in [-9000, -6000) \end{cases}$$

Valores entre [0, 9000] (inclinación hacia la derecha del *smartphone*): El vehículo girará hacia la derecha, para ello, en función de la inclinación el motor derecho perderá relevancia de forma lineal mientras el izquierdo se mantiene al 100%.

$$\begin{cases} Relevancia\ Motor\ Derecho(\%) = 100 & Para\ Roll \in [0{,}500) \\ Relevancia\ Motor\ Derecho(\%) = 100 - \left(\dfrac{Roll \cdot 100}{6000}\right) & Para\ Roll \in [500, 6000] \\ Relevancia\ Motor\ Derecho(\%) = 0 & Para\ Roll \in (6000{,}9000] \end{cases}$$

El resultado de la relevancia de ambos motores en los diferentes rangos de valores del ángulo Roll, se muestra en la siguiente gráfica (Gráfica 1)

ROLL	norm_D_100	norm_I_100
-9000	100	0
-8000	100	0
-7000	100	0
-6000	100	0
-5000	100	17
-4000	100	33
-3000	100	50
-2000	100	67
-1000	100	83
0	100	100
1000	83	100
2000	67	100
3000	50	100
4000	33	100
5000	17	100
6000	0	100
7000	0	100
8000	0	100
9000	0	100



Gráfica 1: Relevancia de los Motores (%) frente al ángulo Roll. Fuente: Propia

- **Pitch:** Determina el "ciclo de trabajo *pitch*" de ambos motores, este ciclo de trabajo posteriormente se verá influenciado por la relevancia de cada motor. Existen 2 posibles casos:

Valores entre [0, 9000] (inclinación hacia adelante del *smartphone*): El vehículo incrementará su velocidad hacia adelante, para ello, en función de la inclinación ambos motores aumentarán su "ciclo de trabaja *pitch*" de forma lineal.

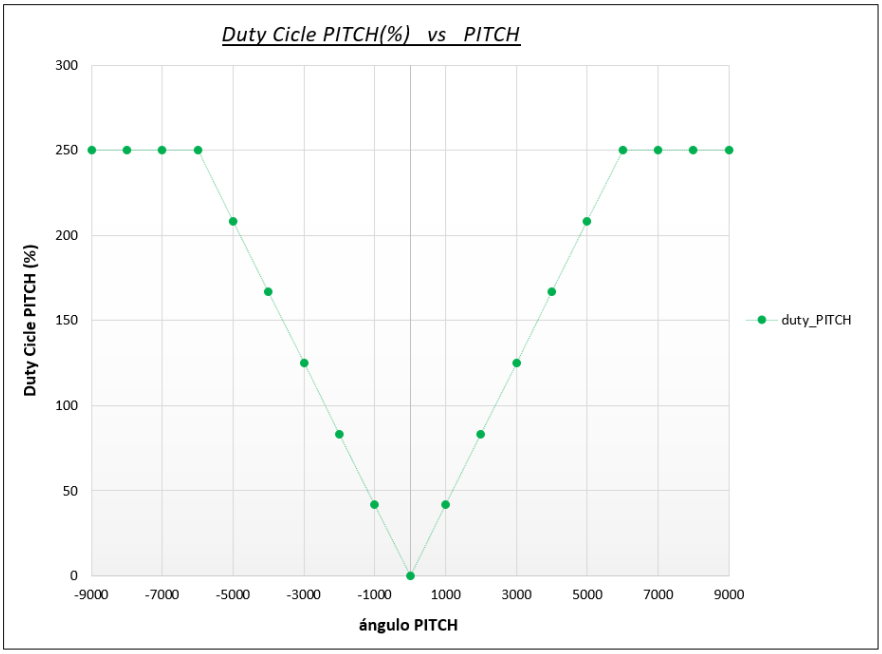
$$\begin{cases} Duty\_Pitch = 0 & Para\ Pitch \in [0, 500) \\ Duty\_Pitch = \dfrac{250 \cdot Pitch}{6000} & Para\ Pitch \in [500, 6000] \\ Duty\_Pitch = 250 & Para\ Pitch \in (6000, 9000] \end{cases}$$

Valores entre [-9000, 0] (inclinación hacia atrás del *smartphone*): El vehículo incrementará su velocidad hacia atrás, para ello, en función de la inclinación ambos motores aumentarán su "ciclo de trabaja *pitch*" de forma lineal.

$$\begin{cases} Duty\_Pitch = 0 & Para\ Pitch \in (-500, 0] \\ Duty\_Pitch = \dfrac{250 \cdot Pitch}{-6000} & Para\ Pitch \in [-6000, -500] \\ Duty\_Pitch = 250 & Para\ Pitch \in [-9000, -6000) \end{cases}$$

El resultado del "ciclo de trabajo pitch" que se aplica a ambos motores en los diferentes rangos de valores del ángulo Pitch, se muestra en la siguiente gráfica (Gráfica 2).

PITCH	duty_PITCH
-9000	250
-8000	250
-7000	250
-6000	250
-5000	208
-4000	167
-3000	125
-2000	83
-1000	42
0	0
1000	42
2000	83
3000	125
4000	167
5000	208
6000	250
7000	250
8000	250
9000	250



Gráfica 2: Duty Cicle PITCH frente al ángulo PITCH. Fuente: Propia

Finalmente, el cálculo de los valores finales de los ciclos de trabajo de los dos motores DC se realiza a través de las siguientes ecuaciones.

$$\boldsymbol{Para\ Pitch\ \geq 0}$$

$$\begin{cases} Duty\ Cicle_{\,Motor\ Derecho} = Duty\_Pitch \cdot \dfrac{Relevancia\ Motor\ Derecho}{100} \\ Duty\ Cicle_{\,Motor\ Izquierdo} = Duty\_Pitch \cdot \dfrac{Relevancia\ Motor\ Izquierdo}{100} \end{cases}$$

$$\boldsymbol{Para\ Pitch\ < 0}$$

$$\begin{cases} Duty\ Cicle_{\,Motor\ Derecho} = 250 - Duty\_Pitch \cdot \dfrac{Relevancia\ Motor\ Derecho}{100} \\ Duty\ Cicle_{\,Motor\ Izquierdo} = 250 - Duty\_Pitch \cdot \dfrac{Relevancia\ Motor\ Izquierdo}{100} \end{cases}$$

Una vez explicado el estudio previo y las ecuaciones que se utilizan para el algoritmo de control, se implemente el código fuente C de la función secundaria "Generar_PWMs" (Figura 99).

```
void Generar_PWMs(float roll,float pitch){
float norm_D_100, norm_I_100, duty_PITCH;

//******* OFFSETS *****
roll = roll -179;
pitch = pitch -179;
//******* ROLL *********
        if(roll >= ROLL_L && roll <= ROLL_H){ // GIRO A DERECHAS REGULADO
        norm_D_100 = 100 - (roll*100/ROLL_H); // Disminución de velocidad en motor Izquierdo
        norm_I_100 = 100;
        }
        else if(roll > ROLL_H){ //GIRO A DERECHAS
        norm_D_100 = 0; // Motor Derecho parado
        norm_I_100 = 100;
        }
        else if(roll <= -ROLL_L && roll >= -ROLL_H){ // GIRO A IZQUIERDAS REGULADO
        norm_D_100 = 100;
        norm_I_100 = 100 - (roll*100/-ROLL_H); // Disminución de velocidad en motor Derecho
        }
        else if(roll < -ROLL_H){ //GIRO A IZQUIERDAS
        norm_D_100 = 100;
        norm_I_100 = 0; // Motor Izquierdo parado
        }
        else{   // SIN GIRO (AMBOS MOTORES AL 100%)
        norm_D_100 = 100;
        norm_I_100 = 100;
        }
//******* PITCH ********
        if(pitch >= PITCH_L && pitch <= PITCH_H){ // HACIA ADELANTE REGULADO
        RC3 = 0; RC0 = 0;   // PWM positiva
        duty_PITCH = (pitch*250/PITCH_H);
        }
        else if(pitch > PITCH_H){ //HACIA ADELANTE (100%)
        RC3 = 0; RC0 = 0;   //PWM positiva
        duty_PITCH = 250;
        }
        else if(pitch <= -PITCH_L && pitch >= -PITCH_H){ //HACIA ATRAS REGULADO
        RC3 = 1; RC0 = 1;   //PWM negativa
        duty_PITCH = (pitch*250/-PITCH_H);
        }
        else if(pitch < -PITCH_H){ //HACIA ATRAS (-100%)
        RC3 = 1; RC0 = 1;   //PWM negativa
        duty_PITCH = 250;
        }
        else{ //SIN MOVIMIENTO (DutyCicle = 0%)
        RC3 = 0; RC0 = 0;   //PWM positiva
        duty_PITCH = 0;
        }
//******* PWMs ********* // Se calculan los respectivos DutyCicles de cada motor
                         // teniendo en cuenta los valores de control calculados
                         // anteriormente por el ROLL y PITCH

    if((pitch >= 0) || (duty_PITCH == 0)){ // CASO HACIA ADELANTE
    CCPR1L = duty_PITCH*(norm_I_100/100); //Duty cicle Motor Izquierdo
    CCPR2L = duty_PITCH*(norm_D_100/100); //Duty cicle Motor Derecho
    }
    else if(pitch<0){   // CASO HACIA ATRAS
    CCPR1L = 250 - (duty_PITCH*norm_I_100/100); //Duty cicle Motor Izquierdo
    CCPR2L = 250 - (duty_PITCH*norm_D_100/100); //Duty cicle Motor Derecho
    }
}
```

```
#define PITCH_H 6000
#define PITCH_L 500
#define ROLL_H 6000
#define ROLL_L 500
```

Figura 99: Código fuente C de la función secundaria Generar_PWMs. Fuente: Propia

9.2.1.4. Función Modo_Manual

Esta función secundaria se aplica cuando el programa se encuentra en el "Modo Libre" (variable *MODO = 'M'*) y tiene como objetivo definir la nueva trayectoria/velocidad del vehículo, a través de los datos recibidos por los inputs accionados desde la "Pantalla del Modo Libre".

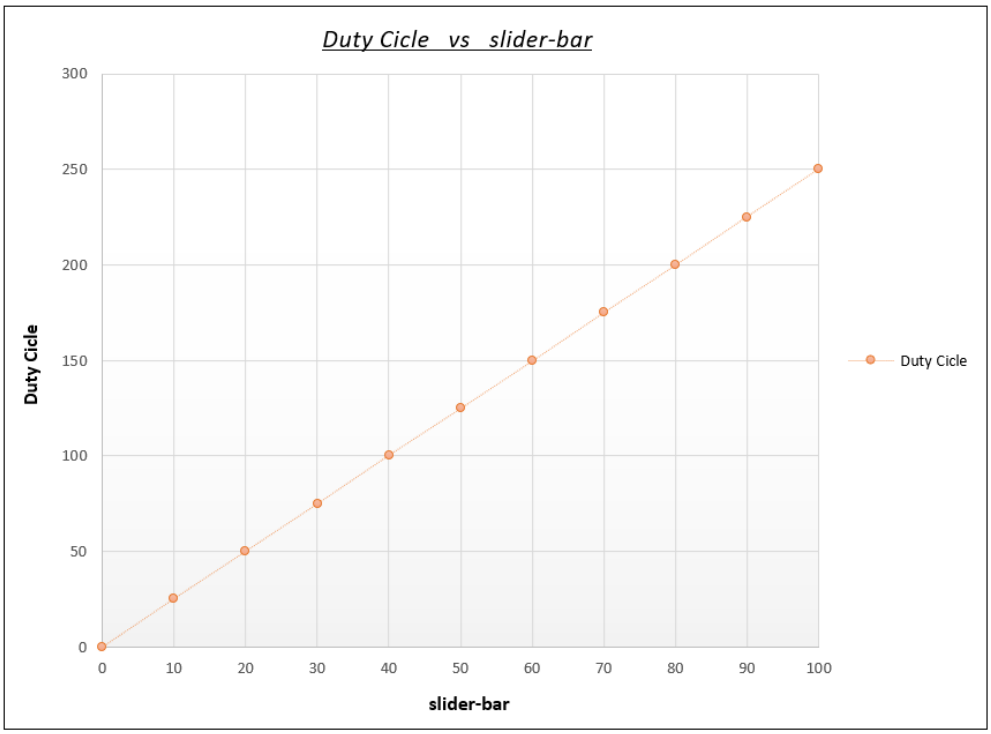
Como se observaba en la Figura 78, se pueden recibir diferentes bytes de datos a través de los diferentes botones de flechas, el botón "STOP" y la *slider-bar*. Por tanto, primeramente se plantean los casos y se decide el control que realizará el algoritmo en este modo de conducción.

- Ciclo de trabajo (*slider-bar*): La *slider-bar* realiza el envío de bytes, datos con números del 0-100 los cuales definirán el ciclo de trabajo de ambos motores. Para ello, dado que el rango del ciclo de trabajo es de [0-250], se aplicará una relación lineal de la forma:

$$y = mx \quad \rightarrow \quad Duty\ Cicle_{250} = 2{,}5 \cdot Duty\ Cicle_{100}$$

El resultado de aplicar esta relación lineal se observa en la gráfica siguiente (Gráfica 3).

SLIDER-BAR	Duty Cicle
0	0
10	25
20	50
30	75
40	100
50	125
60	150
70	175
80	200
90	225
100	250



Gráfica 3:Duty Cicle frente a valores de la slider-bar. Fuente: Propia

- Hacia adelante (letra 't'): Al ser accionada la flecha hacia adelante, a ambos motores se les aplicará el ciclo de trabajo previamente fijado por la *slider-bar*.

Duty Cicle → Motor Derecho (CCPR2L)
Duty Cicle → Motor Izquierdo (CCPR1L)
} caída de tensión positiva (+)

- Hacia atrás (letra 'u'): Al ser accionada la flecha hacia atrás, a ambos motores se les aplicará el ciclo de trabajo previamente fijado por la *slider-bar*. En este caso, se ha de aplicar de forma inversa, debido a que la caída de tensión en los motores es negativa y el sentido de giro de las ruedas contrario.

250 - Duty Cicle → Motor Derecho (CCPR2L)
250 - Duty Cicle → Motor Izquierdo (CCPR1L)
} caída de tensión negativa (-)

- Giro a derechas (letra 'v'): Accionando la flecha hacia la derecha, solo se ha de aplicar el ciclo de trabajo (prefijado por la *slider*-bar) al motor izquierdo dado que el motor derecho ha de estar parado para que se produzca el giro. Se diferencian 2 casos distintos, en función de si la caída de tensión en los motores es positiva (hacia adelante) o negativa (hacia atrás).

0 → Motor Derecho (CCPR2L)
Duty Cicle → Motor Izquierdo (CCPR1L)
} caída de tensión positiva (+)

---

250 → Motor Derecho (CCPR2L)
250 - Duty Cicle → Motor Izquierdo (CCPR1L)
} caída de tensión negativa (-)

- Giro a izquierdas (letra 'w'): Accionando la flecha hacia la izquierda, solo se ha de aplicar el ciclo de trabajo (prefijado por la *slider*-bar) al motor derecho dado que el motor izquierdo ha de estar parado para que se produzca el giro. Se igual modo se diferencian 2 casos en función del signo de la tensión en los bornes de los motores DC.

Duty Cicle → Motor Derecho (CCPR2L)
0 → Motor Izquierdo (CCPR1L)
} caída de tensión positiva (+)

---

250 – Duty Cicle → Motor Derecho (CCPR2L)
250 → Motor Izquierdo (CCPR1L)
} caída de tensión negativa (-)

❖ STOP (letra ‘z’): Accionando el botón “STOP”, Se aplica un ciclo de trabajo nulo a ambos motores, además de establecer una caída de tensión positiva en estos.

0 → Motor Derecho (CCPR2L)

0 → Motor Izquierdo (CCPR1L)

} caída de tensión positiva (+)

Una vez explicada la funcionalidad de cada caso, se aplica el código fuente correspondiente para aplicar el control de este modo (Figura 100).

```
void Modo_Manual(int dato){ // Calcula la trayectoria y velocidad del
                            // vehiculo en funcion del dato recibido
int DUTY_CCP;               // (de la slider-bar o de los botones)

    if(dato > 100)  letra_M = dato; // Datos de botones "letras"
    else    duty_cicle_M = dato;    // Datos de la slider-bar "0-100"

    DUTY_CCP = (duty_cicle_M*5)/2;  // Se adaptan los datos al rango de
                                    // valores del DutyCicle (0-250)

    switch(letra_M){ // TRAYECTORIAS EN FUNCION DE LAS LETRAS (t u v w, z "STOP")

        case 't':   //COCHE HACIA ADELANTE
            RC3 = 0; RC0 = 0;   //PWM positiva
            CCPR1L = DUTY_CCP;
            CCPR2L = DUTY_CCP; // A ambos motores se le aplica el DutyCicle
        break;                 // de la slider-bar

        case 'u':   //COCHE HACIA ATRÁS
            RC3 = 1; RC0 = 1;   //PWM negativa
            CCPR1L = 250 - DUTY_CCP;
            CCPR2L = 250 - DUTY_CCP; // A ambos motores se le aplica el DutyCicle
        break;                       // de la slider-bar de forma inversa (-)

        case 'v':   //COCHE HACIA LA DERECHA

            if(RC0 == 0){ // CASO HACIA ADELANTE

                CCPR1L = DUTY_CCP;
                CCPR2L = 0; // Motor Derecho parado
            }else if (RC0 == 1){ // CASO HACIA ATRAS
                CCPR1L = 250 - DUTY_CCP;
                CCPR2L = 250; // Motor Derecho parado
            }
        break;

        case 'w':   //COCHE HACIA LA IZQUIERDA

            if(RC0 == 0){ // CASO HACIA ADELANTE

                CCPR1L = 0; // Motor Izquierdo parado
                CCPR2L = DUTY_CCP;
            }else if (RC0 == 1){ // CASO HACIA ATRAS
                CCPR1L = 250; // Motor Izquierdo parado
                CCPR2L = 250 - DUTY_CCP;
            }
        break;

        case 'z':   //COCHE PARADO
            RC3 = 0; RC0 = 0;   //PWM positiva
            CCPR1L = 0;
            CCPR2L = 0; //Ambos Motores PARADOS
        break;
    }
}
```

Figura 100: Código fuente C de la función secundaria Modo_Manual. Fuente: Propia

9.2.2. Interrupción por "ECHO" del sensor de ultrasonidos (RBIF)

La interrupción "*ECHO"* realiza la detención del vehículo cuando la distancia entre el sensor ultrasonidos que incorpora en la parte delante del chasis y un objeto es inferior a 10,2 cm. Además, hay que tener en cuenta que solo se realiza en el modo de conducción "Libre".

La activación del flanco de interrupción RBIF se produce en el momento que se inicia el pulso de la señal "*ECHO*" (flanco de subida) y una vez que termina (flanco de bajada) pudiendo calcular la distancia entre vehículo-objeto a partir de la duración de éste.

En el siguiente diagrama de bloques se observa las diferentes acciones que se aplican durante la atención a esta interrupción.

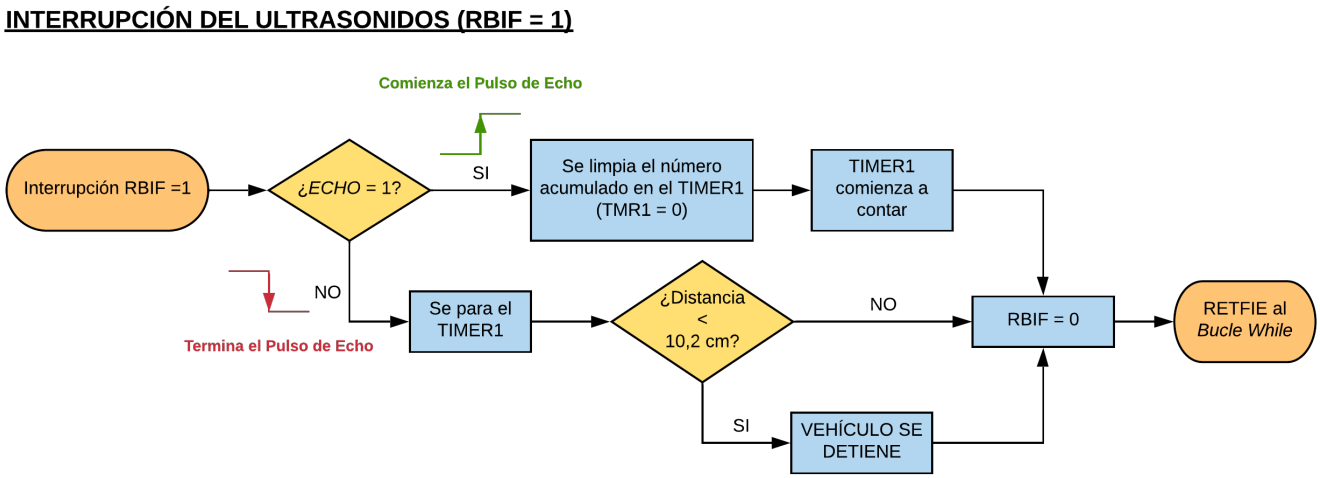


Figura 101: Diagrama de Bloques de la Interrupción RBIF. Fuente: Propia

El TIMER1 comienza a contar con el inicio del pulso de "*ECHO*" y se detiene una vez ha finalizado. Para la configuración del TIMER 1, se ha seguido las explicaciones referentes al apartado 6.9, pudiendo verse el resultado del registro T1CON en el código fuente de la función *SETUP* (Figura 86) .

Configuración del TIMER1 (T1CON):

- Fuente de CLK = f_{OSC} /4
- Pre-Scaler = 8
- Oscilador externo (T1OSC) apagado
- TIMER1 apagado (TMR1ON = 0)

Con la configuración establecida, el tiempo equivalente al número de pulsos contados por el TIMER1 se rige por la siguiente ecuación (Ecuación 7):

$$Tiempo_{TMR1} = N_{TMR1} \cdot 8 \cdot \frac{4}{16 \cdot 10^6}$$

Ecuación 7: Tiempo equivalente del TIMER1 como contador de pulsos (fosc/4). Fuente: Propia

Finalmente, para calcular la distancia a partir del tiempo que ha permanecido el pulso "*ECHO*" en valor lógico alto, se introduce esta nueva ecuación (Ecuación 7) en la Ecuación 6 dada por el fabricante [6] quedando la siguiente expresión (Ecuación 8):

$$Distancia\ (cm) = \left(N_{TMR1} \cdot 8 \cdot \frac{4}{16 \cdot 10^6}\right) \cdot \frac{340\ m/s}{2} \cdot \frac{100\ cm}{1\ m}$$

Ecuación 8: Distancia en cm detectada por el sensor ultrasonidos a partir del tiempo medido por el TIMER1. Fuente: Propia

Dado que la finalidad de esta interrupción es que se detenga el vehículo para una distancia con un objeto inferior a 10,2 cm, el valor del TMR1 que condicionará esta parada será calculado despejando en la Ecuación 8 el término N_{TMR1}.

$$N_{TMR1} = 10{,}2cm \cdot \frac{1}{8} \cdot \frac{16 \cdot 10^6}{4} \cdot \frac{2}{34000\ cm/s} = 300$$

Una vez explicado el método seguido, se implementa el código fuente correspondiente (Figura 102).

```
else if (INTCONbits.RBIF){ // INTERRUPCIÓN DEL ECHO DEL ULTRASONIDOS

    switch(echo){

        case 1: //Comienza el pulso de ECHO
            TMR1=0;
            TMR1ON=1; // El TIMER1 comienza a contar
        break;

        case 0: //Termina el pulso de ECHO

            TMR1ON=0; // El TIMER1 se detiene
                      // Temp_TMR1(us) = N*8*0,25us

            if(TMR1<300){ //Menos de 10,2cm (Dist_cm = Temp_TMR1/2*340*100)
            CCPR1L=0;CCPR2L=0; // Si la distancia es menor que 10,2 cm se para
            }
        break;
    }
    RBIF=0;  //Desactivamos la bandera de recepción de las int por cambio de flanco en el PORTB
}
```

Figura 102: Código fuente de la interrupción RBIF. Fuente: Propia

10. Implementación

En este apartado de la memoria se detallará y explicará la metodología seguida para el desarrollo y construcción del vehículo de tres ruedas paso a paso.

Durante el desarrollo del proyecto, se han ido siguiendo una serie de pasos intermedios con el fin de realizar diferentes comprobaciones del funcionamiento de los módulos utilizados. Para ello, se desarrollaron distintos programas en código fuente C, de los cuales algunas partes han sido reutilizadas en el programa final del proyecto.

La implementación del vehículo puede dividirse en 3 partes:

- La primera parte desarrolla el montaje del vehículo a partir de las piezas del kit comprado para el proyecto.
- La segunda parte alcanza la comprobación del correcto funcionamiento de los distintos módulos conectados al PIC, estos son, el módulo *Bluetooth*, el módulo L293D de amplificación de corriente para los motores y el sensor de ultrasonidos.
- Finalmente la tercera parte consistirá en realizar la implementación del control del vehículo mediante los modos “Libre” y “Giroscopio”. Además, se realizarán diferentes mediciones para asegurar que el control se realiza de forma correcta y eficiente.

10.1. Montaje de las piezas del KIT del vehículo

El primer paso del proyecto consistió en realizar el montaje del chasis del vehículo de tres ruedas para poder situar todos los elementos y módulos que forman el proyecto en él.

Las piezas que componen el KIT del vehículo se pueden observar en la siguiente imagen (Figura 103)

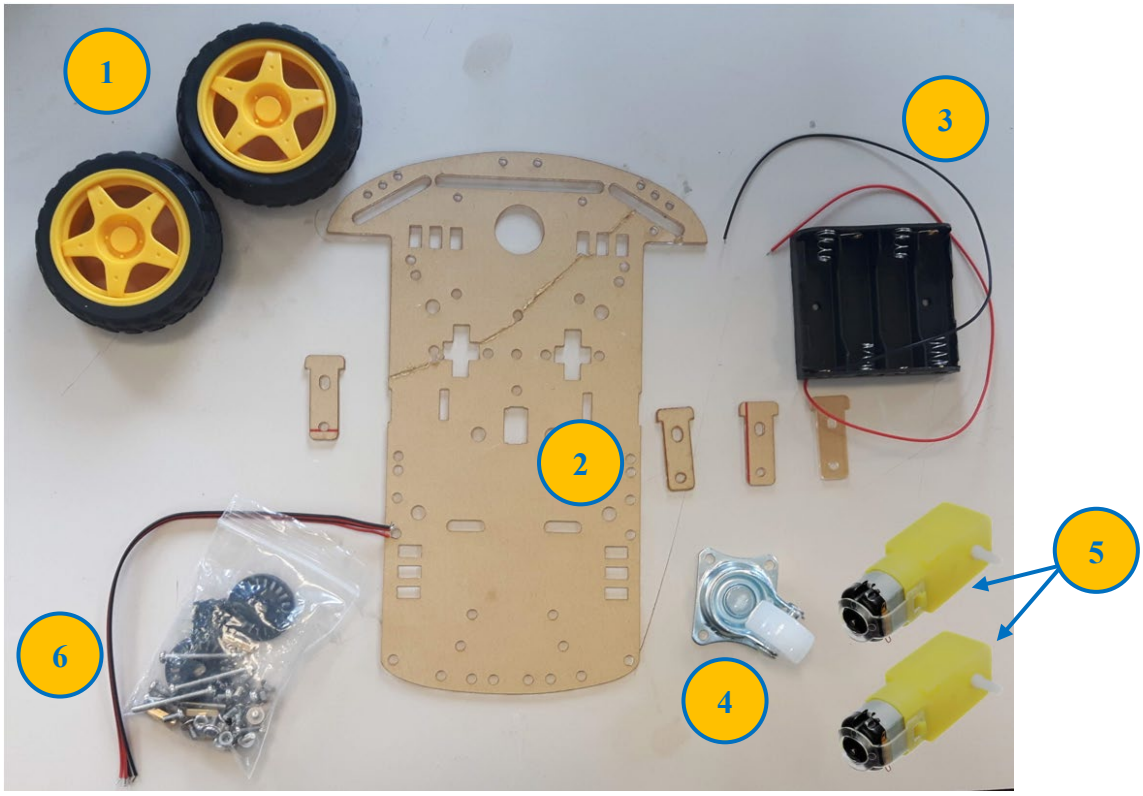


Figura 103: Piezas del KIT del vehículo de 3 ruedas. Fuente: Propia

A continuación se listan las diferentes partes del KIT de la Figura 103:

1. Ruedas delanteras del vehículo
2. Chasis de plástico y soportes para los motores
3. Soporte para las pilas (4 pilas AA)
4. Rueda trasera del vehículo
5. Motores DC
6. Piezas para la transmisión del vehículo, tornillos, arandelas y otros

Una vez realizada la descripción de las diferentes partes, se procede al montaje del vehículo de tres ruedas añadiendo también una "*Protoboard*" en la parte superior de este, no incluida en el KIT. El resultado final del vehículo con todas las piezas montadas se muestra a continuación (Figura 104).

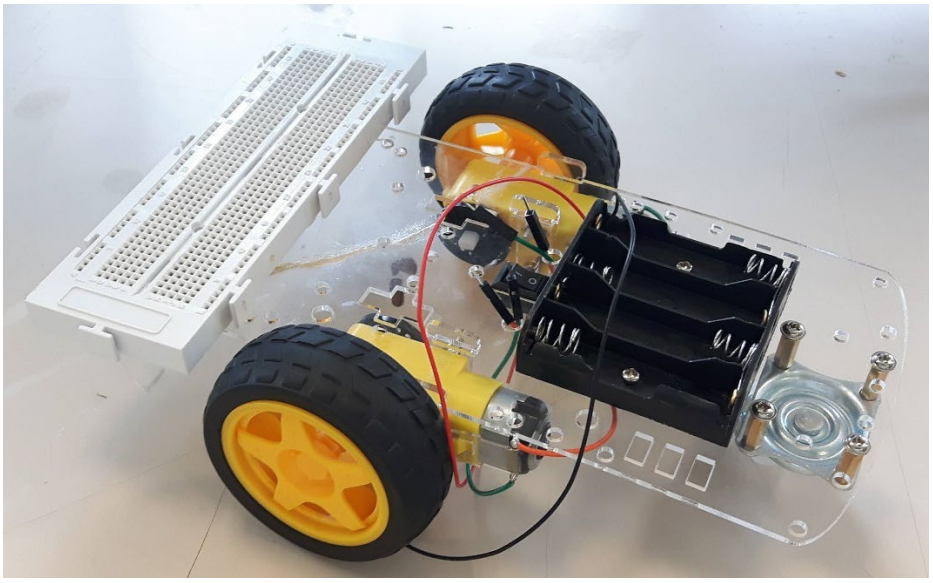


Figura 104: Resultado del montaje del KIT del vehículo de 3 ruedas. Fuente: Propia

10.2. Pruebas de los Periféricos Externos

En este proyecto los periféricos externos o módulos que implementa el vehículo, son clave para captar y reaccionar a los diferentes estímulos que se producen en el entorno que rodea el coche. Debido a ello, se ha de entender muy bien su funcionamiento y asegurarse de que la programación que los controla es la correcta.

Los distintos periféricos sobre los que se realizaron pruebas fueron:

- Módulo *Bluetooth* HC-05
- Módulo L293D Mini Motor Shield
- Sensor de Ultrasonidos HC-SR04

En los siguientes subapartados se explicarán y expondrán los resultados de las pruebas de cada uno de los periféricos listados.

10.2.1. Pruebas con Módulo Bluetooth HC-05

Primeramente, se realizaron las pruebas correspondientes en el módulo *Bluetooth* HC-05. Para ello se creó una aplicación la cuál podía iniciar la comunicación Bluetooth y enviar datos con valores del 0-50 mediante una *slider-bar.*

Para poder visualizar el resultado de la prueba, se conectó la sonda del osciloscopio en el pin TX del módulo *BT* (conectado con el pin RX del PIC), se puede observar esta conexión en la siguiente figura (Figura 105).

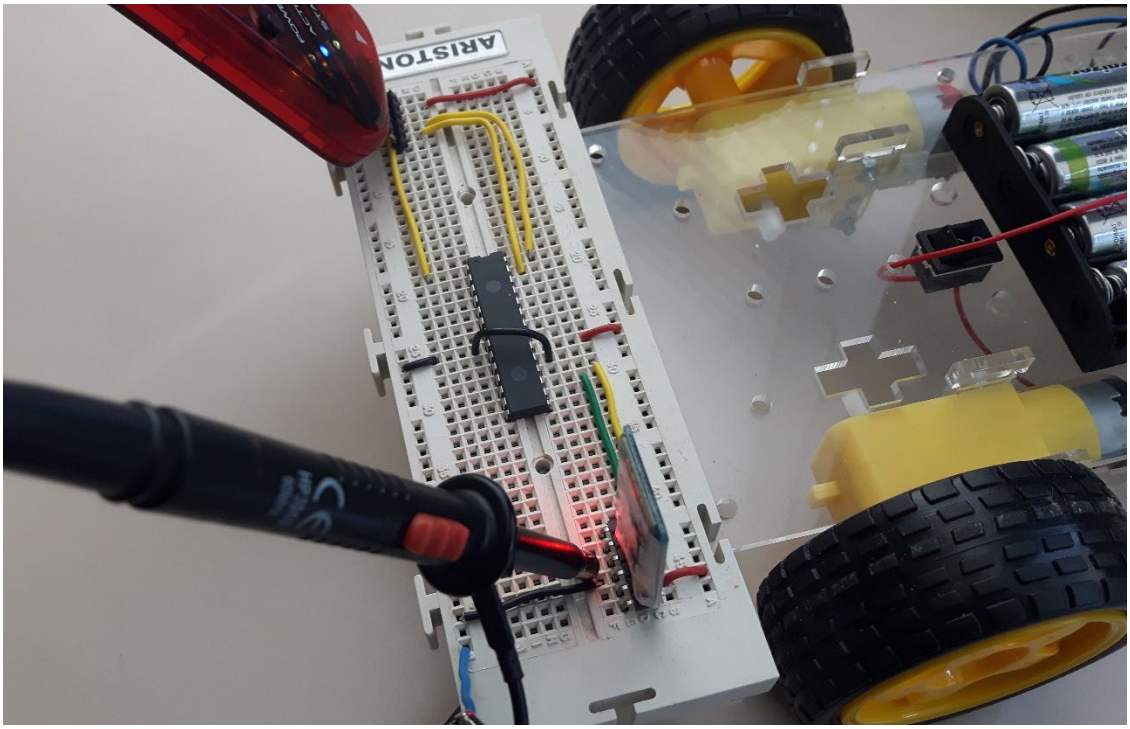


Figura 105: Conexión de la sonda del osciloscopio en el pin TX del módulo Bluetooth HC-05. Fuente: Propia

Una vez realizada las conexiones correspondientes e iniciada la conexión *Bluetooth* entre el teléfono *smartphone* y el módulo, atendiendo a lo explicado en el apartado 6.11.2, se realiza el envío de diferentes datos.

Además, previamente al envío de estos datos se ha de configurar de forma correcta el módulo AUSART del PIC. Esta configuración se realizó a partir del fichero “uart.h” explicado anteriormente (apartado 9.2.1.1).

➢ **<u>Envío del dato "42":</u>** En binario el valor decimal 42 es *0010 1010*. Se eligió este número dado que su trama de datos es fácil de localizar.

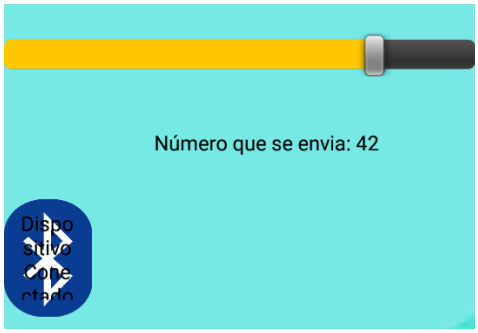


Figura 106: Envío del valor decimal 42 mediante la app en la prueba del módulo BT. Fuente: Propia

El resultado recibido en el pin RX del PIC tras el envío del dato mediante la aplicación móvil (Figura 106), es el siguiente:

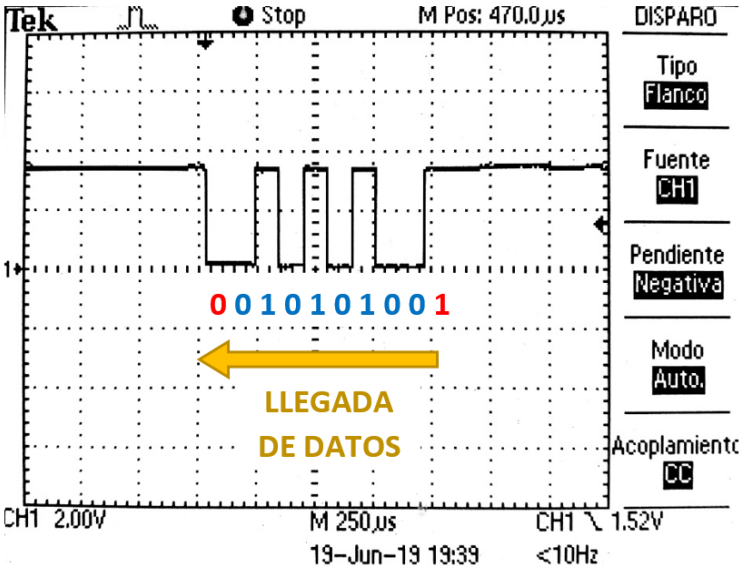


Figura 107: Trama de datos del valor 42 recibida en el pin RX del PIC. Fuente: Propia

El envío de la trama de datos comienza con el bit de inicio ("*bit Start*"), el cual tiene valor lógico bajo "0",tras él se envía el byte de datos <u>empezando por el bit 0</u> y <u>terminando por el bit 7</u> tal que:

b0 b1 b2 b3 b4 b5 b6 b7 → 0 1 0 1 0 1 0 0

La lectura de los bits se ha de hacer de izquierda a derecha, de esta forma se obtiene el dato 42:

b7 b6 b5 b4 b3 b2 b1 b0 → 0 0 1 0 1 0 1 0 $_{Bin}$= 42 $_{Dec}$

Finalmente, se recibe el bit de parada ("*bit Stop*") cuyo valor lógico es alto "1"

➢ **<u>Envío del dato "17":</u>** De igual forma, se realiza el envío del valor decimal 17 en decimal *0001 0001*.

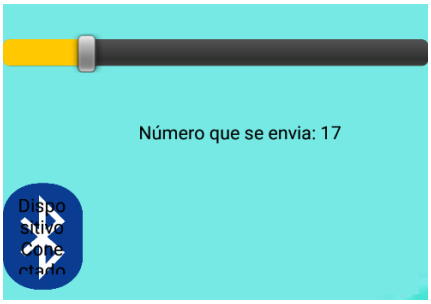


Figura 108: Envío del valor decimal 17 mediante la app en la prueba del módulo BT. Fuente: Propia

Una vez enviado el dato mediante la aplicación móvil (Figura 108), en el pin RX del PIC se recibe el siguiente resultado (Figura 109):

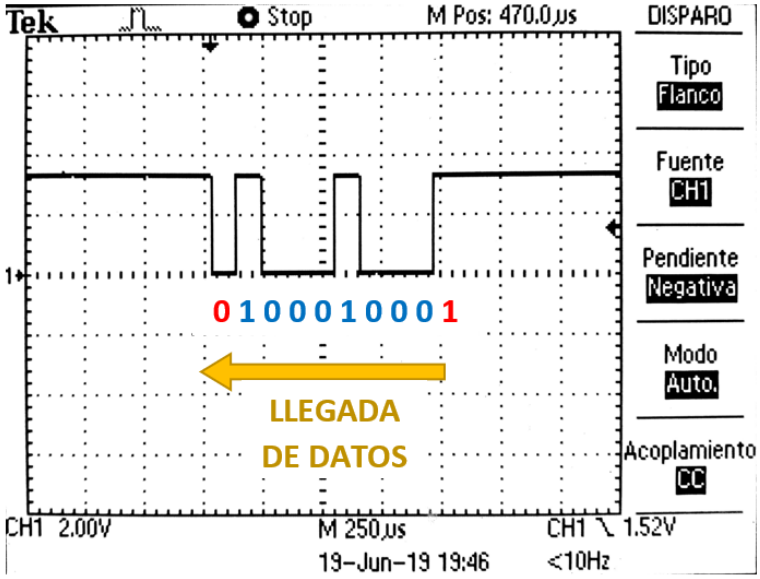


Figura 109: Trama de datos del valor 17 recibida en el pin RX del PIC. Fuente: Propia

Se inicia la llegada de la trama de datos con el bit de inicio ("*bit Start*", "0"), posteriormente se recibe el byte de datos (b0-b7):

b0 b1 b2 b3 b4 b5 b6 b7 → 1 0 0 0 1 0 0 0

La lectura de los bits se ha de hacer de izquierda a derecha, de esta forma se obtiene el dato 17:

b7 b6 b5 b4 b3 b2 b1 b0 → 0 0 0 1 0 0 0 1 $_{Bin}$= 17 $_{Dec}$

Finalmente, se recibe el bit de parada ("*bit Stop*", "1")

Se puede concluir que el resultado de esta prueba fue el correcto. Además, observando la escala de tiempo de ambas figuras (Figura 107 y Figura 109), cada bit de datos tiene una duración de 125us. Por lo tanto, dado que se envían 2 bits de inicio/final y 8 bits del dato, la duración de cada trama completa es de $^{125us}/_{bit} \cdot 10\,^{bit}/_{trama} = 1{,}25\ ms/trama$.

10.2.2. Pruebas con el Módulo L293D Mini Motor Shield

Para poder realizar el movimiento controlado de ambos motores DC del vehículo, se han de generar dos señales PWM a 1000Hz mediante el microcontrolador PIC, que posteriormente serán amplificadas por el módulo L293D.

Para la toma de resultados de la prueba, se utiliza una aplicación capaz de establecer la comunicación *Bluetooth* entre el teléfono *smartphone* y el vehículo, además de enviar valores del 0-250 mediante una *slider-bar* (Este programa fue utilizado posteriormente para la creación de la pantalla "Modo Libre").

Las conexiones de los aparatos de medida se realizan antes y después de la amplificación de una sola señal PWM realizada por el módulo L293D. En la zona previa a la amplificación se utiliza la sonda del osciloscopio para ver la forma de la señal generada por el módulo PWM. Una vez la señal es amplificada, se mida mediante un amperímetro la corriente que se le suministra a cada motor. Estas conexiones pueden apreciar en la figura mostrada a continuación (Figura 110).

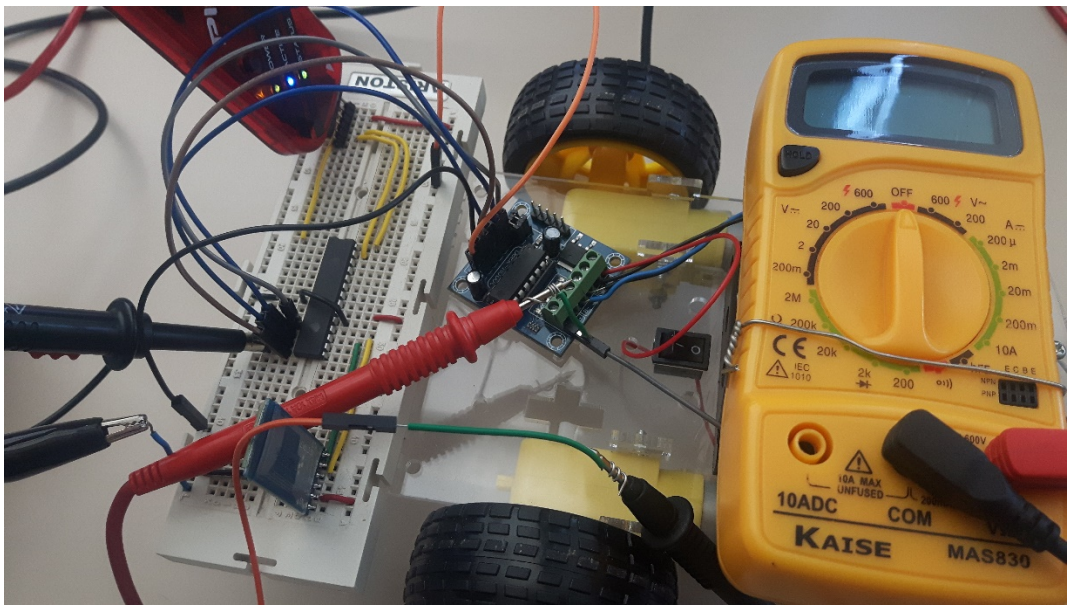


Figura 110: Conexiones de la sonda del osciloscopio y el amperímetro en las etapa previa y posterior a la amplificación con el módulo L293D. Fuente: Propia

- PWM con Ciclo de trabajo al 25%: La primera prueba se realiza desplazando la *slider-bar* hasta una posición cercana a su 25%. Estos valores enviados van de 0-250 (mismos valores que el CCPRxL), lo cual significa que se envía un valor decimal de 63 que corresponde con un ciclo de trabajo de la PWM de 25%.

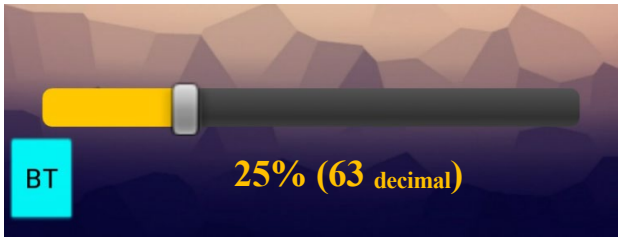


Figura 111: Envío del valor decimal 63 mediante la app en la prueba del módulo L293D. Fuente: Propia

Una vez es realizado el envío del dato, mediante el programa cargado en el PIC, se genera la señal PWM correspondiente en su pin CCP1 (pin 13, RC2). El resultado de la medida de esta señal con la sonda del osciloscopio es la siguiente (Figura 112).

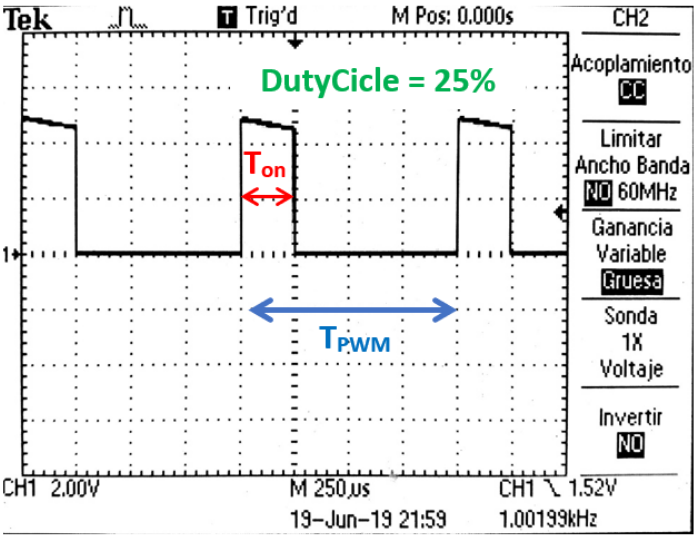


Figura 112: Señal PWM al 25% del ciclo de trabajo generada por el pin CCP1. Fuente: Propia

Finalmente, se mide la intensidad de corriente que se suministra a los motores con el ciclo de trabajo a 25% tras la etapa de amplificación (Figura 113).

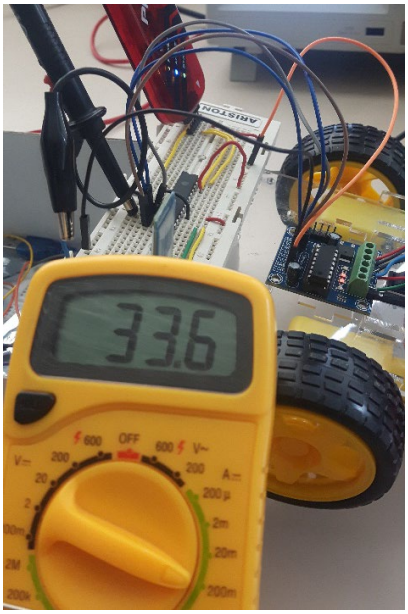


Figura 113: Corriente suministrada a los motores (mA) con ciclo de trabajo de la PWM a 25%. Fuente: Propia

La corriente suministrada a cada motor DC es de 33,6 mA (Escala del amperímetro a 200 mA).

- PWM con Ciclo de trabajo al 75%: Se desplaza la *slider-bar* hasta una posición cercana a su 75%. El dato enviado mediante la comunicación *Bluetooth* será correspondiente al valor decimal de 188 que corresponde con un ciclo de trabajo de la PWM de 75%.

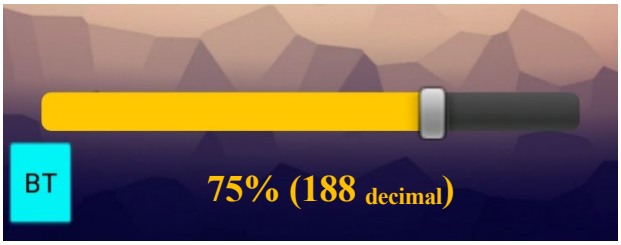


Figura 114: Envío del valor decimal 188 mediante la app en la prueba del módulo L293D. Fuente: Propia

Tras recibir correctamente el dato, el microcontrolador PIC genera la señal PWM de salida en su pin CCP1. El resultado de la medida de esta señal con la sonda del osciloscopio es la siguiente (Figura 115).

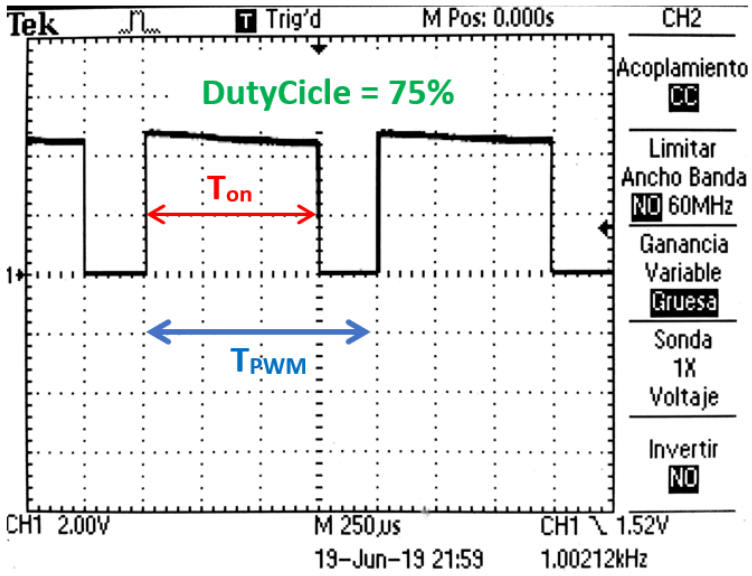


Figura 115: Señal PWM al 75% del ciclo de trabajo generada por el pin CCP1. Fuente: Propia

De igual forma, se realiza la medida de la intensidad de corriente que suministra el módulo L293D a los motores con el ciclo de trabajo a 75% (Figura 116).

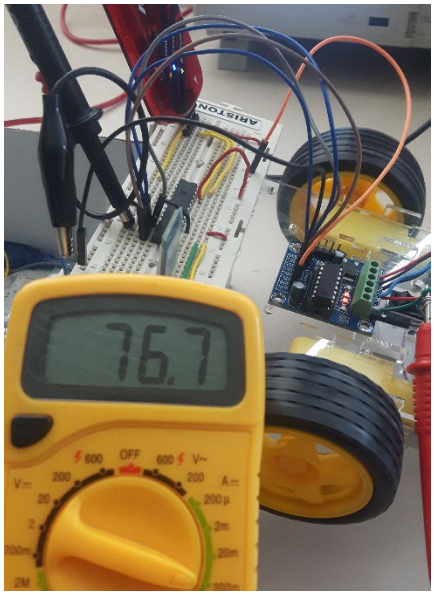


Figura 116: Corriente suministrada a los motores (mA) con ciclo de trabajo de la PWM a 75%. Fuente: Propia

La corriente suministrada a cada motor DC es de 76,7 mA (Escala del amperímetro a 200 mA).

- PWM con Ciclo de trabajo al 100%: Finalmente para comprobar el caso extremo, donde la velocidad del vehículo es máxima, e desplaza la *slider-bar* hasta su posición máxima (100%). Como resultado, la trama de datos enviada portará el valor decimal 250. Además, la lectura de este valor realizará la posterior generación de una señal PWM con ciclo de trabajo de 100% por la salida del pin CCP1 del microcontrolador.

Figura 117: Envío del valor decimal 250 mediante la app en la prueba del módulo L293D. Fuente: Propia

La señal generada por el PIC tras recibir el dato es la siguiente (Figura 118):

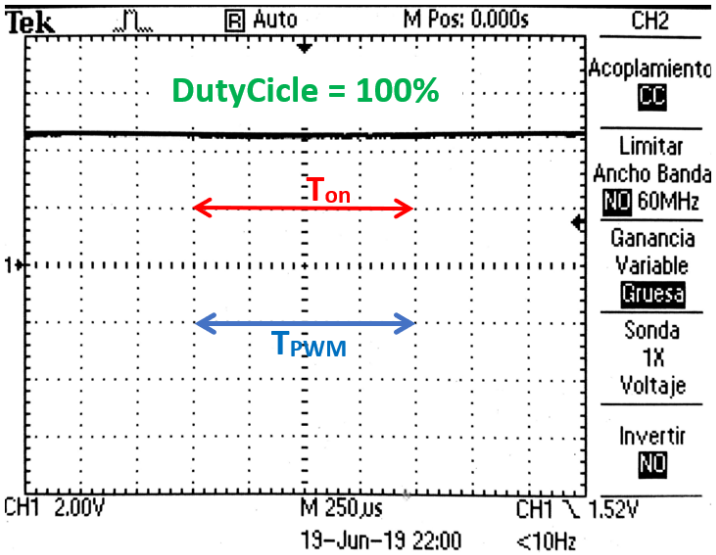


Figura 118: Señal PWM al 100% del ciclo de trabajo generada por el pin CCP1. Fuente: Propia

La señal observada en la Figura 118, es una señal continua a 5V, dado que el ciclo de trabajo es del 100% y por tanto no hay pulsos en valor lógico bajo "0V" ($T_{PWM}=T_{on}$). Además, para este caso se obtendrá la corriente de salida máxima que suministra el módulo L293D a los motores DC (Figura 119).

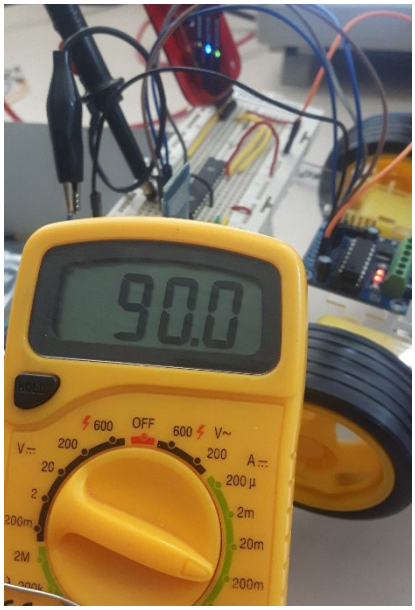


Figura 119: Corriente suministrada a los motores (mA) con ciclo de trabajo de la PWM a 100%. Fuente: Propia

La corriente máxima de salida del módulo L293D tiene un valor medio de 90mA (Figura 119), aunque se observaron pequeñas oscilaciones de este valor de ± 1,2 mA.

Tras las pruebas realizadas se puede concluir que la utilización de este módulo de amplificación es decisiva para el buen funcionamiento del vehículo, dado que consigue ampliar la corriente suministrada por un pin I/O del microcontrolador de 25mA hasta los 90mA (Aumento de un 360%).

10.2.3. Pruebas con el Sensor de Ultrasonidos

El sensor de ultrasonidos tiene como objetivo evitar colisiones entre el vehículo y los obstáculos su entorno. Previamente a la realización de las pruebas se estudió el funcionamiento de este, explicado en el apartado 7.4 de esta memoria.

Para poder realizar las pruebas, se realizó un programa de código fuente C que fue cargado en el microcontrolador. En este programa únicamente se generaba el pulso de "*trigger*" de anchura 10us para poder ver la señal de salida "*echo*" generada por el sensor. El resultado del pulso de "*trigger*" generado puede observarse en la Figura 120.

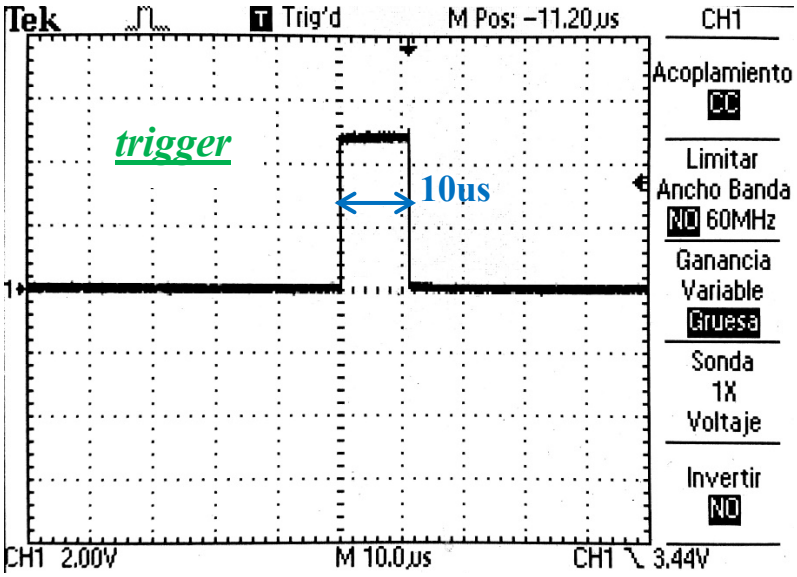


Figura 120: Pulso de la señal trigger de 10us generado por el PIC. Fuente: Propia

En la medición de ambas señales ("*echo*" y "*trigger*") se utilizan las dos sondas del osciloscopio. Las conexiones utilizadas para la toma de medidas se muestran en la figura siguiente (Figura 121).

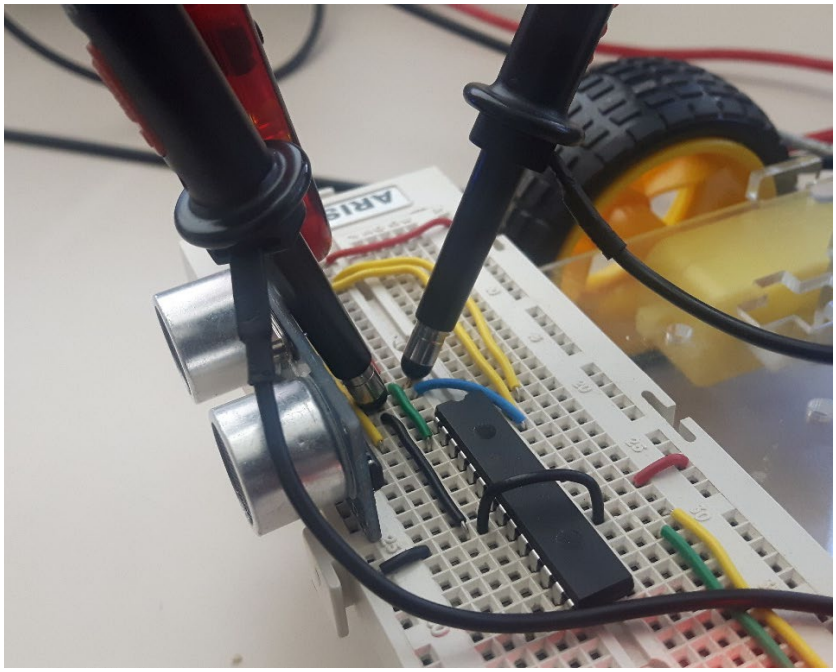


Figura 121: Conexión de las sondas del osciloscopio con las señales trigger y echo del sensor de ultrasonidos. Fuente: Propia

Una vez realizado el montaje, se realizan 2 mediciones a diferentes distancias con el sensor de ultrasonidos. Para ello se posiciona un objeto dentro de su rango de visión.

- Distancia al objeto de 5 cm: Primeramente, se situó el objeto a una distancia de 5cm. De esta forma, la ráfaga de ultrasonidos generada por el sensor consigue rebotar en el obstáculo y retornar generando la señal de salida "*echo*". La anchura de esta señal es equivalente al tiempo que ha tardado en su ida y vuelta.

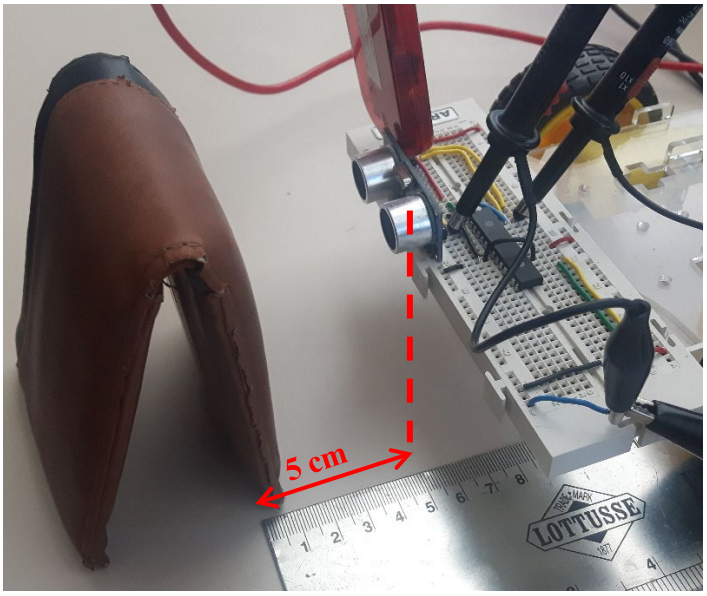


Figura 122: Objeto a una distancia de 4cm del sensor de ultrasonidos. Fuente: Propia

El resultado de las señales "*trigger*" y "*echo*" medidas mediante las sondas del osciloscopio es el siguiente (Figura 123).

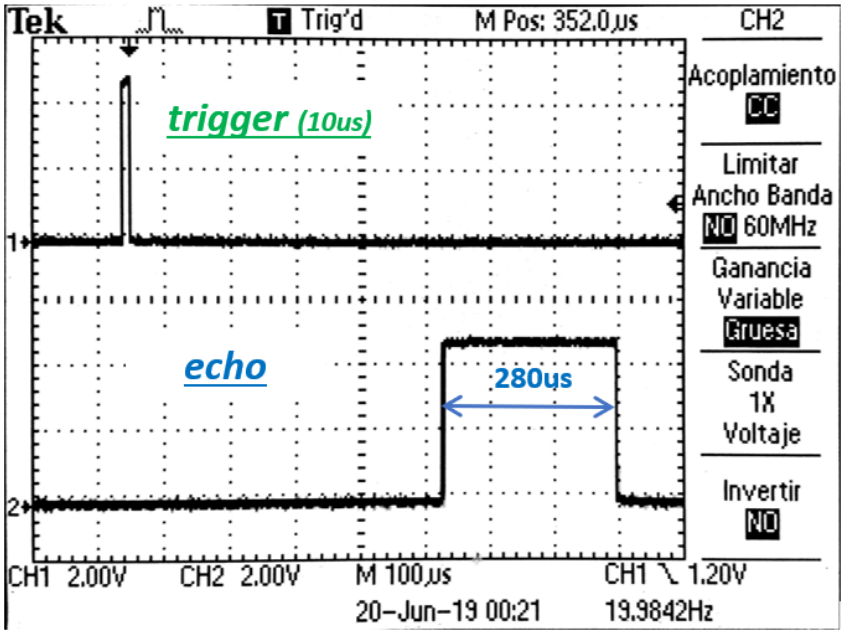


Figura 123: Señales trigger y echo con un objeto a 5cm del sensor de ultrasonidos. Fuente: Propia

La duración del pulso de "*echo*" medido es de 280us. Mediante la Ecuación 6 se obtiene la distancia en cm equivalente.

$$Distancia(cm) = 280us \cdot \frac{10^{-6}s}{us} \cdot \frac{34000\frac{cm}{s}}{2} = 4{,}76\ cm$$

Finalmente se obtiene la distancia de 4,76 cm, la cual se ajusta al valor esperado teniendo en cuenta que las mediciones y la forma del objeto presentan errores de precisión.

- Distancia al objeto de 10 cm: En esta última prueba, se situó el objeto a 10 cm de distancia con el sensor de ultrasonidos (). Además, esta distancia es la que implementa el código fuente C en el programa final.

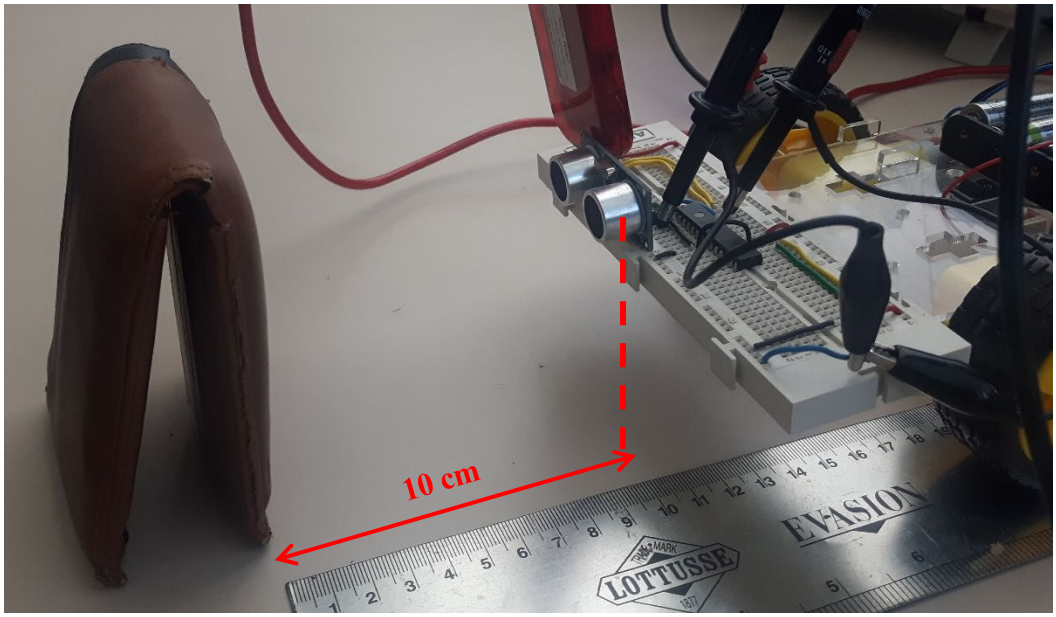


Figura 124: Objeto a una distancia de 10cm del sensor de ultrasonidos. Fuente: Propia

El resultado de las señales "*trigger*" y "*echo*" medidas mediante las sondas del osciloscopio se muestra a continuación (Figura 125).

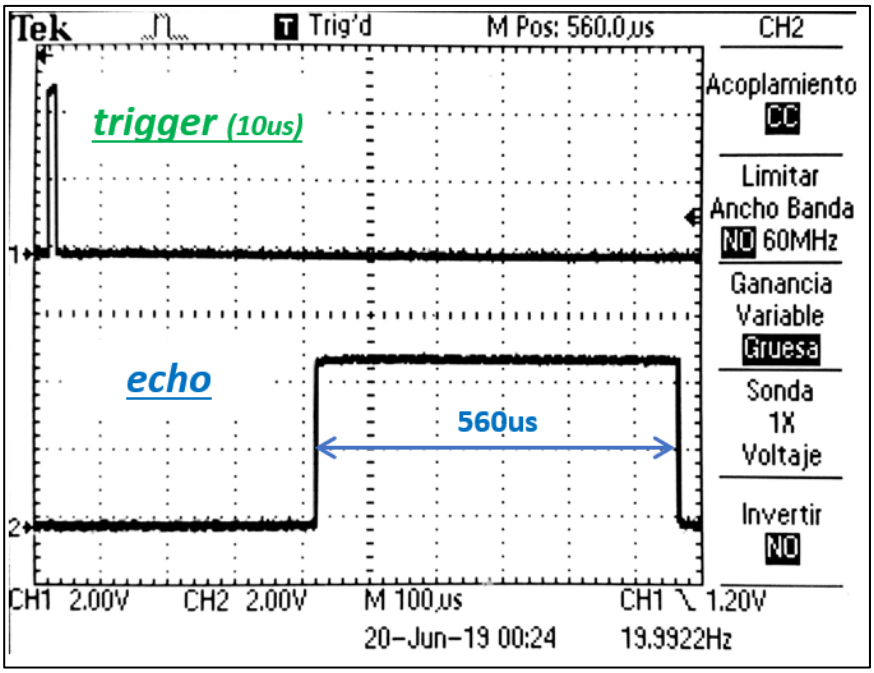


Figura 125: Señales trigger y echo con un objeto a 10cm del sensor de ultrasonidos. Fuente: Propia

La duración del pulso de "*echo*" medido es de 560us. De igual forma que en la prueba anterior, utilizando la Ecuación 6 se obtiene la distancia en cm equivalente.

$$Distancia(cm) = 560us \cdot \frac{10^{-6}s}{us} \cdot \frac{34000\frac{cm}{s}}{2} = 9{,}52\ cm$$

La distancia calculada es de 9,52 cm, esta pequeña desviación en la precisión de la medida (0,48 cm) es debido a los errores anteriormente comentados en el caso de 5cm.

10.3. Pruebas en los modos de control del vehículo

En este apartado se explicarán las últimas pruebas realizadas del proyecto respecto a los modos de conducción que implementa el vehículo.

Para la toma de resultados de ambos, antes de tener la aplicación para *smartphone* terminada, se realizaron por separado las diferentes pantallas de "Modo Libre" y "Modo Giroscopio". De esta forma, se pudo asegurar previamente el buen funcionamiento de ambas partes y reutilizar las pantallas realizadas para la última versión de la aplicación móvil.

10.3.1. Pruebas del Modo Libre

Esta primera parte de pruebas tiene como objetivo visualizar las diferentes PWM generadas durante el modo de conducción "Libre". Estas señales varían en función de la trayectoria y la velocidad seleccionada por el usuario mediante los inputs de los botones y la *slider-bar*.

Para visualizar los resultados de las señales de ambos motores DC, se conectan las dos sondas del osciloscopio a los pines CCP1 y CCP2 del microcontrolador (Etapa previa a la amplificación del módulo L293D). Estas conexiones se muestran en la siguiente figura (Figura 126).

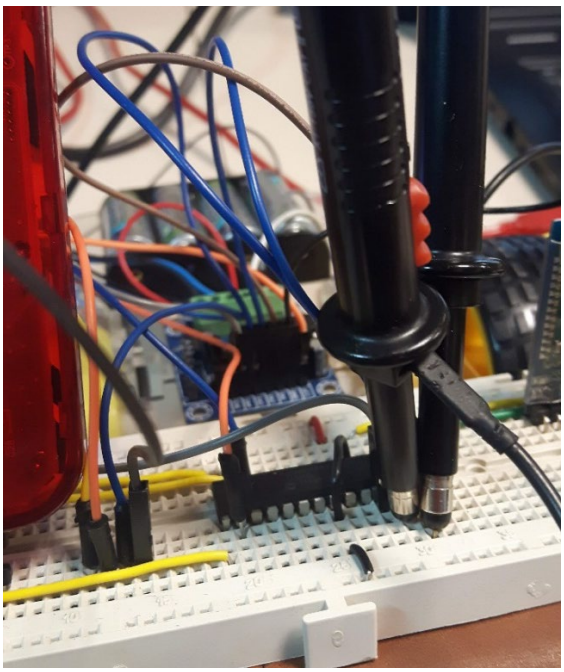


Figura 126: Conexión de las sondas del osciloscopio con las señales PWM de los pines CCP1 y CCP2. Fuente: Propia

Una vez realizadas las conexiones de ambas sondas, se realizan diferentes pruebas para las distintas interacciones del usuario con los inputs de esta la pantalla.

- **<u>Movimiento hacia adelante (Flecha ↑):</u>** Dejando la *slider-bar* en su posición por defecto al iniciar la pantalla del "Modo Libre" (50% de recorrido) se acciona el botón de la flecha hacia arriba. Como resultado, a ambos motores se les aplicará una PWM con caída de tensión positiva y ciclo de trabajo al 50% (Figura 127).

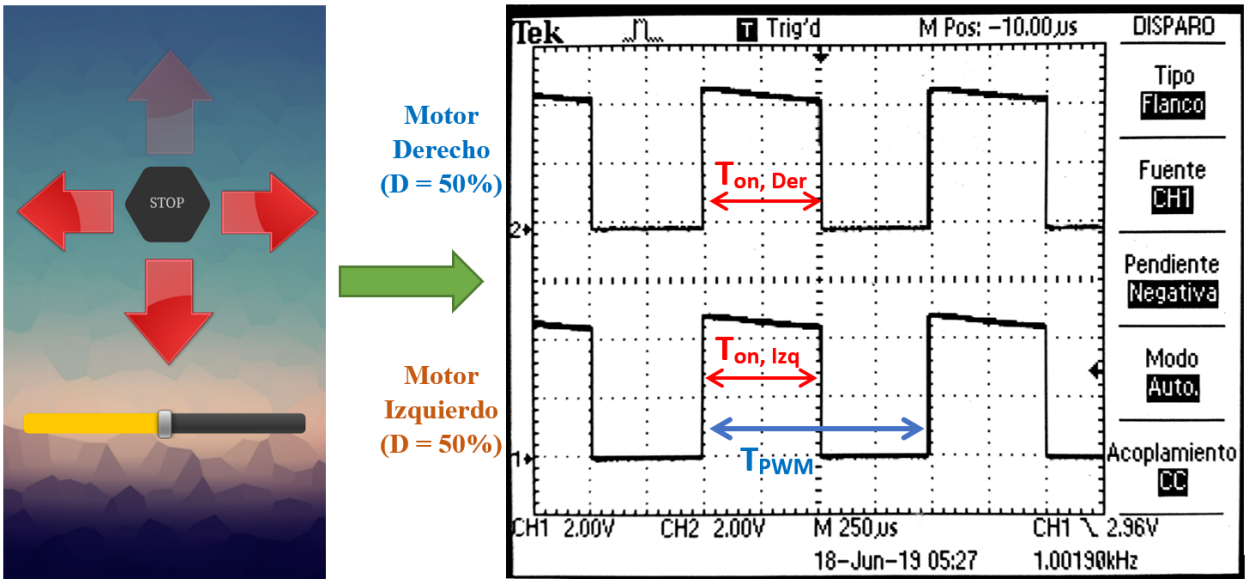


Figura 127: Accionamiento de botón hacia arriba con 50% ciclo de trabajo y resultado de las PWM en los pines CCP1 y CCP2 del microcontrolador. Fuente: Propia

Una vez realizado la prueba con la *slider-bar* en su posición por defecto, se interactúa con ella hasta al alcanzar un valor de ciclo de trabajo del 75% y poder visualizar como varían las señales PWM suministradas a ambos motores (Figura 128).

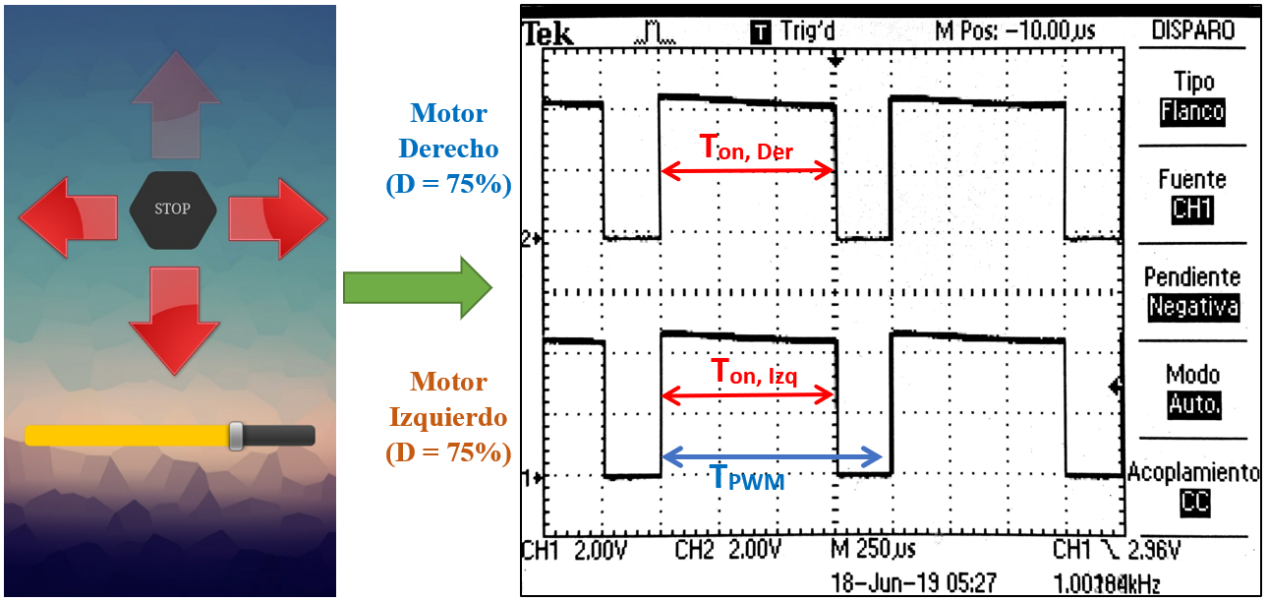


Figura 128: Accionamiento de botón hacia arriba con 75% ciclo de trabajo y resultado de las PWM en los pines CCP1 y CCP2 del microcontrolador. Fuente: Propia

- **<u>Movimiento hacia atrás (Flecha ↓):</u>** Situando la *slider-bar* al 75% en la pantalla del "Modo Libre", se acciona el botón de la flecha hacia atrás. En este caso, a los dos motores DC se les aplica las señales PWM con caída de tensión negativa y ciclo de trabajo al 75% (Figura 129).

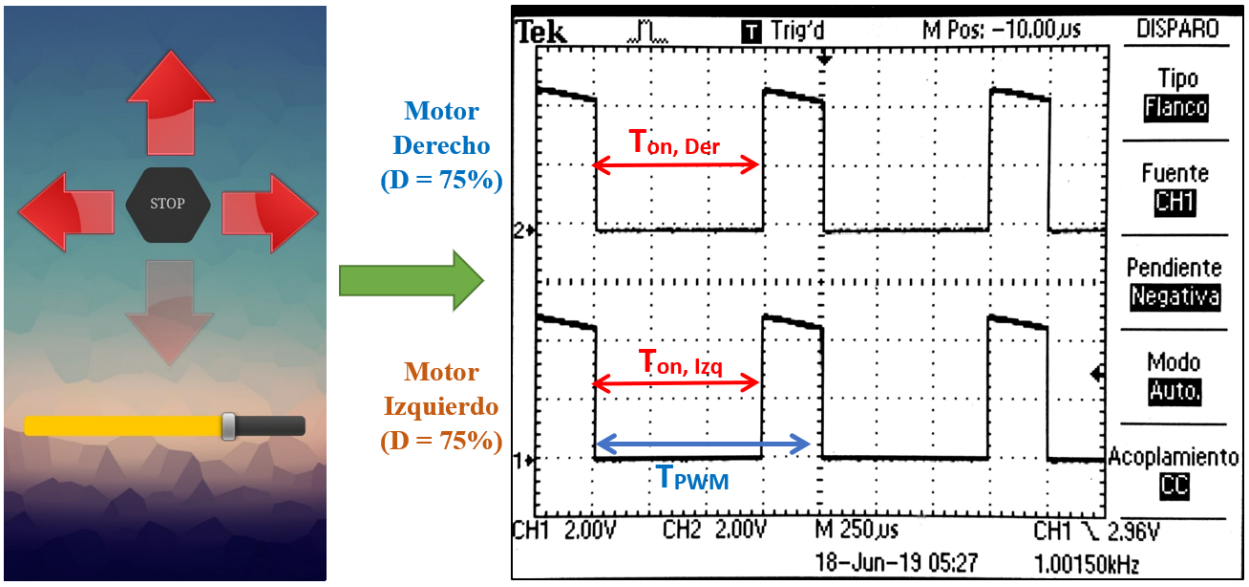


Figura 129: Accionamiento de botón hacia abajo con 75% ciclo de trabajo y resultado de las PWM en los pines CCP1 y CCP2 del microcontrolador. Fuente: Propia

- **<u>Giro hacia la derecha (Flecha →):</u>** Situando nuevamente la *slider-bar* a la mitad de su posición (50%), se acciona el botón de giro a la derecha. Debido a ello, el ciclo de trabajo de la señal PWM del motor derecho será "0%", mientras que en el motor izquierdo se aplicará el ciclo de trabajo definido por la *slider-bar* (en este caso un 50%).

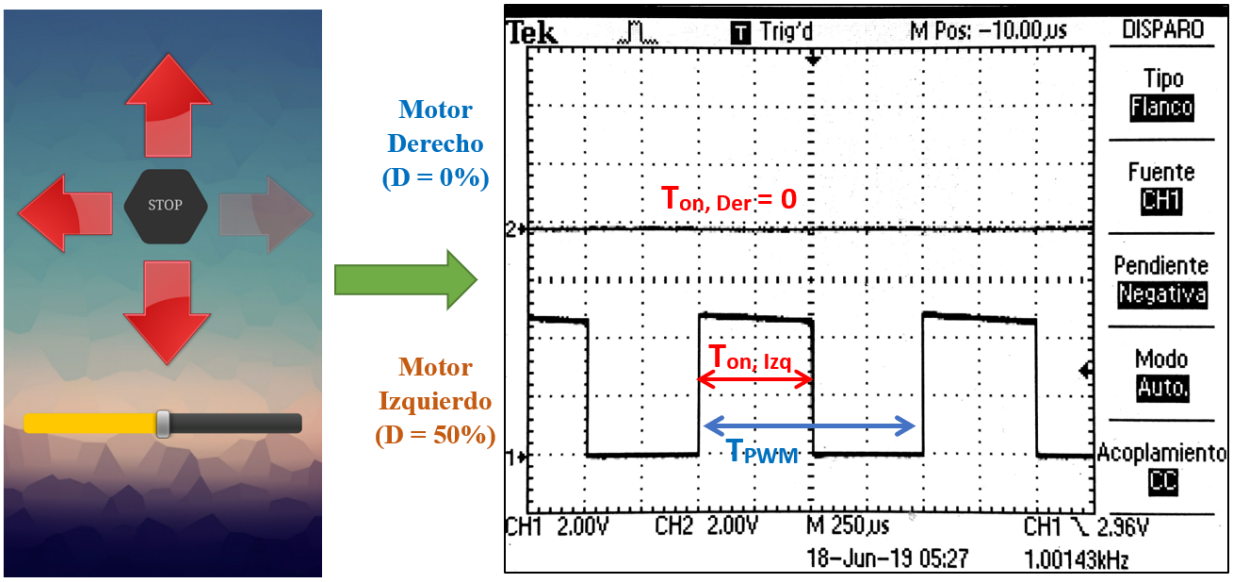


Figura 130: Accionamiento de botón hacia la derecha con 50% ciclo de trabajo y resultado de las PWM en los pines CCP1 y CCP2 del microcontrolador. Fuente: Propia

- **Movimiento hacia la izquierda (Flecha ←):** De igual forma que en la prueba anterior (giro a la derecha), manteniendo el ciclo de trabajo al 50% se acciona el botón de giro a la izquierda. En este caso el resultado es el inverso, con el ciclo de trabajo nulo en el motor izquierdo (Figura 131).

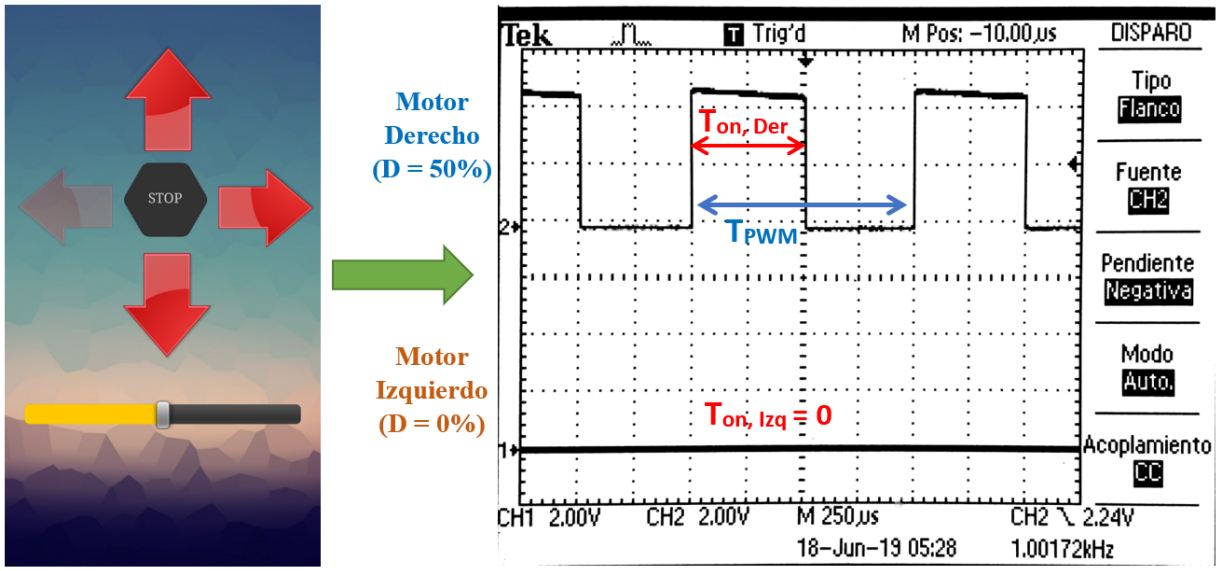


Figura 131: Accionamiento de botón hacia la izquierda con 50% ciclo de trabajo y resultado de las PWM en los pines CCP1 y CCP2 del microcontrolador. Fuente: Propia

- **Detención del vehículo (botón STOP):** Finalmente se realiza la parada de ambos motores DC pulsando el botón "STOP" de la pantalla. En esta situación, la posición de la *slider-bar* no es relevante dado que se aplica a las señales PWM de ambos motores un ciclo de trabajo de 0% (Figura 132).

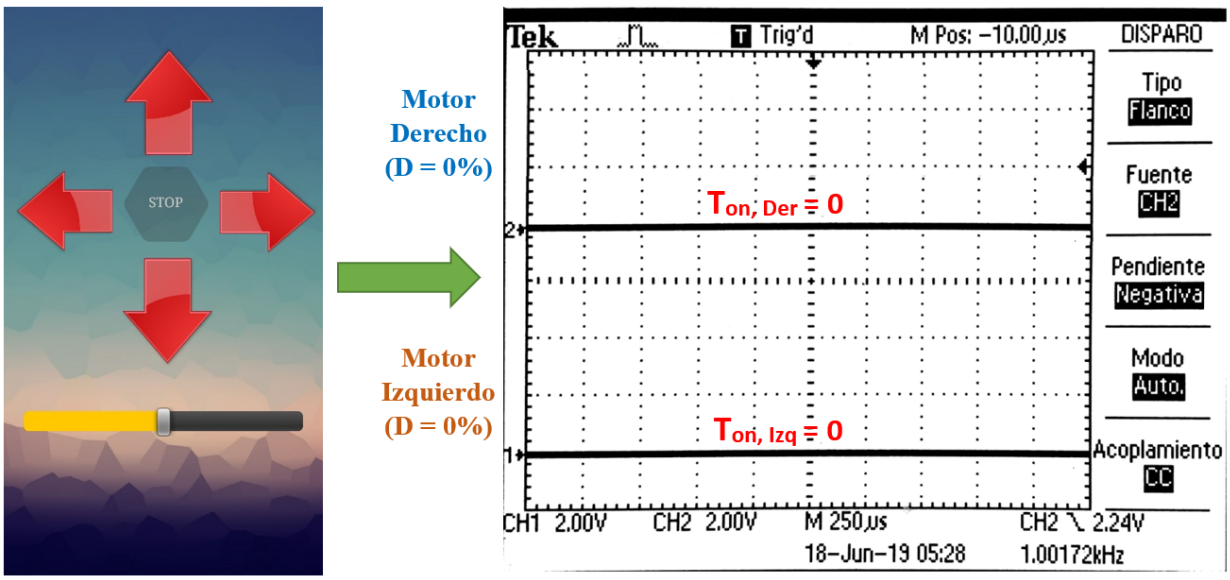


Figura 132: Accionamiento de botón STOP y resultado de las PWM en los pines CCP1 y CCP2 del microcontrolador. Fuente: Propia

10.3.2. Pruebas del Modo Giroscopio

Las últimas pruebas del "Modo Giroscopio" se centran en visualizar las señales PWM generadas por las diferentes inclinaciones del giroscopio tipo MEMS del teléfono *smartphone*.

Respecto a las conexiones realizadas para visualizar ambas señales PWM suministradas a los motores, se mantienen las realizadas en las pruebas del "Modo Libre" mostradas en la Figura 126. Además, dado que en este modo de conducción existen multitud de opciones de movimiento y trayectoria, solo se van a explicar las situaciones consideradas más relevantes.

- **<u>Inclinación hacia DELANTE:</u>** En esta primera prueba se realiza la inclinación del teléfono *smartphone* únicamente hacia adelante (Pitch > 0 ; Roll ≈ 0). En la siguiente figura (Figura 133), se puede apreciar la inclinación adquirida por el dispositivo *Android* durante el desarrollo de la prueba.

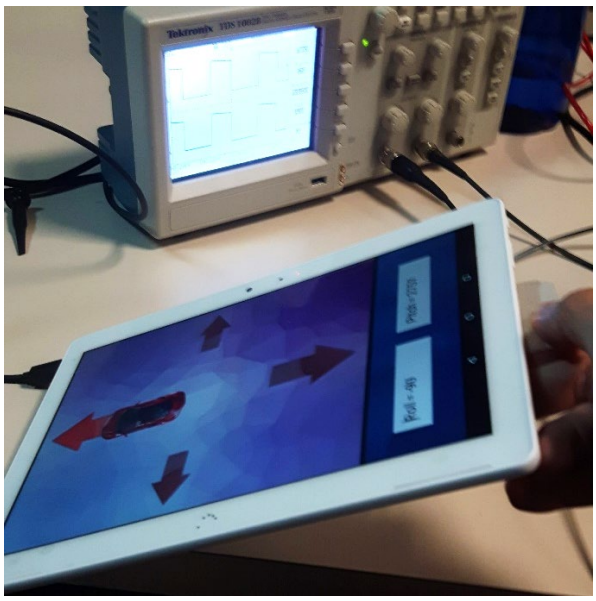


Figura 133: Inclinación del dispositivo smartphone hacia Delante en la prueba del Modo Giroscopio. Fuente: Propia

Tras realizar el cambio en la orientación del giroscopio tipo MEMS del dispositivo, se envían los datos Roll/Pitch al microprocesador. Una vez son recibidos, se interpretan y ejecutan las acciones correspondientes gracias al programa cargado en PIC.

En este caso dado que la inclinación del Roll no es significativa, ambos motores tendrán relevancia para realizar una trayectoria recta. Por otro lado, el Pitch se encargará principalmente de ajusta la velocidad del vehículo. Los resultados de esta prueba se muestran a continuación (Figura 134).

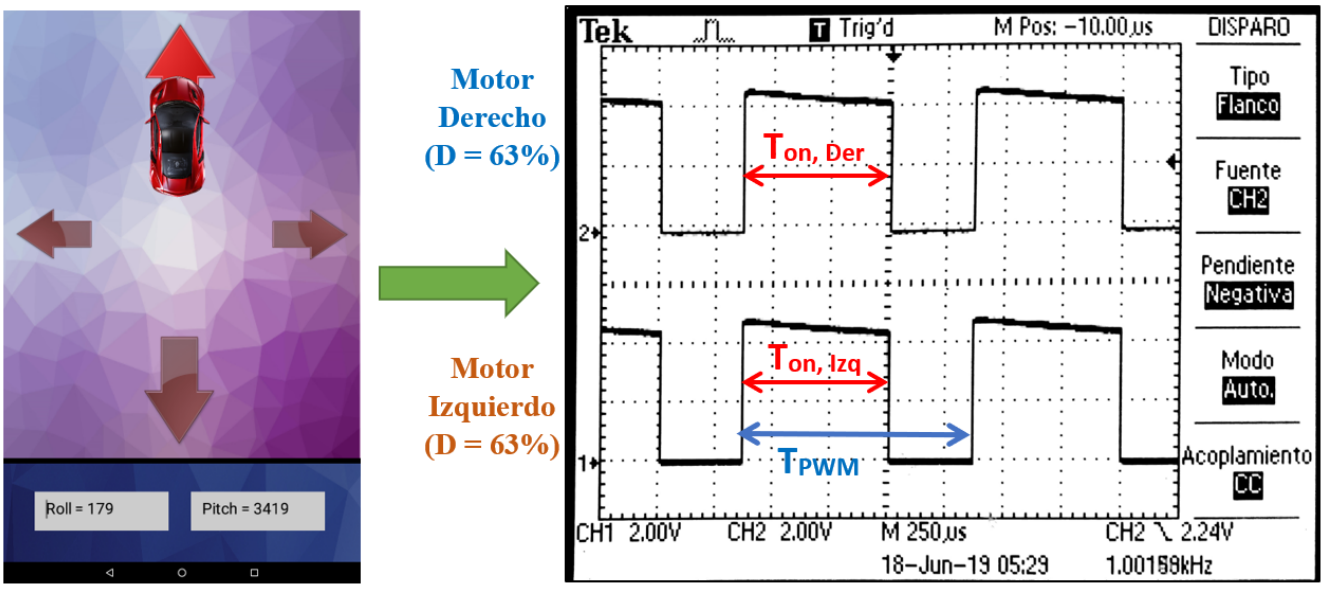


Figura 134: Inclinación en la aplicación móvil hacia Delante (Roll = 1,79º ; Pitch = 34,19º) y resultado de las PWM en los pines CCP1 y CCP2 del microcontrolador. Fuente: Propia

El resultado obtenido en la Figura 134 es el esperado. Ambos motores tienen la misma relevancia (Roll ≈ 0) por lo que el ciclo de trabajo determinado por el Pitch es similar en los dos (63%).

- **Inclinación hacia DELANTE/DERECHA:** En la siguiente prueba se realiza la inclinación del teléfono *smartphone* tanto hacia adelante (Pitch > 0) como hacia la derecha (Roll > 0) para un caso concreto de los muchos que pueden darse. La orientación adoptada por el dispositivo *Android* se observa a continuación (Figura 135).

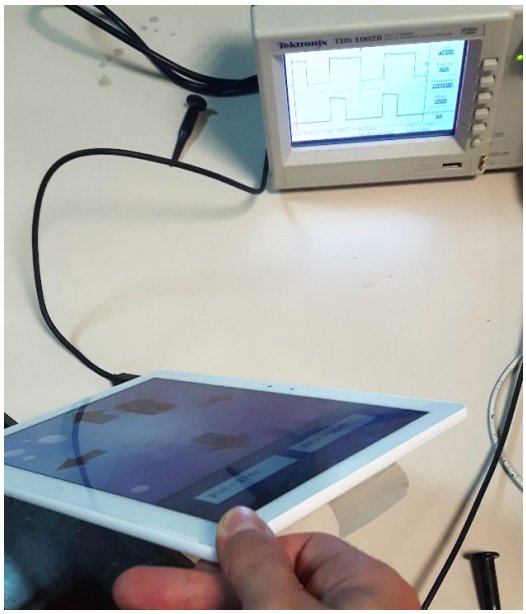


Figura 135 Inclinación del dispositivo smartphone hacia Delante y Derecha en la prueba del Modo Giroscopio. Fuente: Propia

Al inclinar el dispositivo hacia la derecha, el motor izquierdo obtiene mayor relevancia respecto al derecho , como se explicaba en el apartado 9.2.1.3 de esta memoria. De esta forma, el vehículo gira hacia la derecha en función del ángulo de Roll realizado y de manera lineal. La toma de resultados es la siguiente (Figura 136).

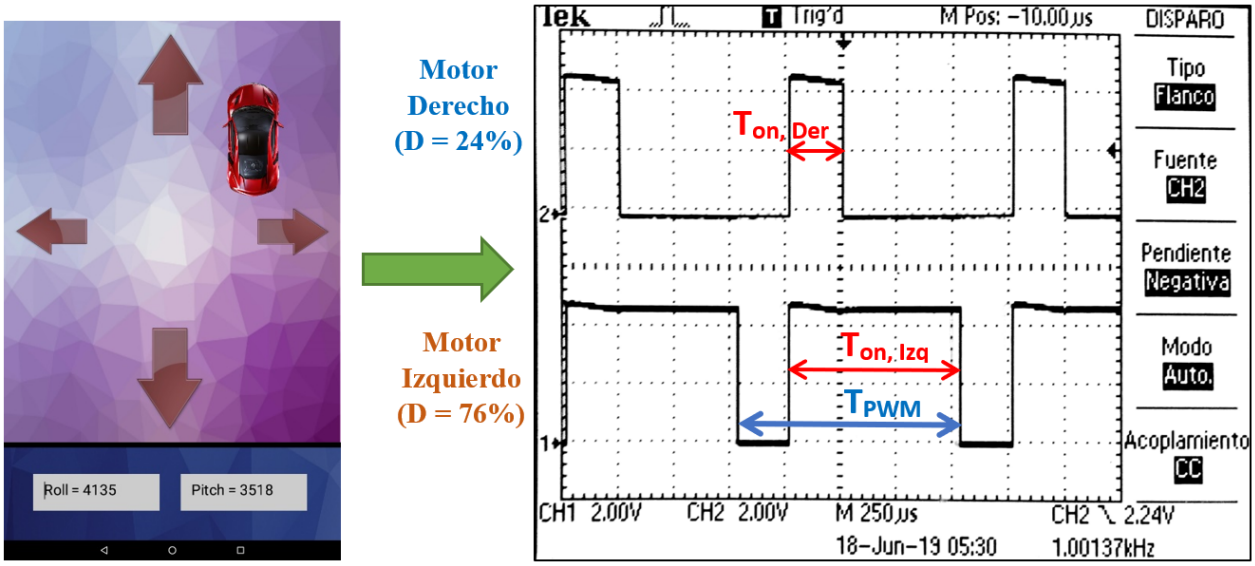


Figura 136: Inclinación en la aplicación móvil hacia Delante y Derecha (Roll = 41,35° ; Pitch = 35,18°) y resultado de las PWM en los pines CCP1 y CCP2 del microcontrolador. Fuente: Propia

El ángulo Pitch adoptado otorgaría a ambos motores el mismo ciclo de trabajo, sin embargo, al estar presente un ángulo de Roll positivo de 41,35º la relevancia del motor derecho disminuye hasta un 32% frente al 100% del motor izquierdo.

$$DutyCicle_{Der} = DutyPitch \cdot \frac{32}{100} = 76\% \cdot 0{,}32 = \mathbf{24\%}$$

$$DutyCicle_{Izq} = DutyPitch \cdot \frac{100}{100} = 76\% \cdot 1 = \mathbf{76\%}$$

- **<u>Inclinación hacia DELANTE/IZQUIERDA:</u>** De igual forma que en la prueba anterior, se realiza la inclinación del dispositivo tanto hacia delante (Pitch > 0) como hacia la izquierda (Roll < 0). Los resultados mostrados a continuación hacen referencia a un solo caso de los muchos que se podrían adoptar con este tipo de inclinación. La orientación del *smartphone* durante la prueba fue la siguiente (Figura 137).

Figura 137: Inclinación del dispositivo smartphone hacia Delante e Izquierda en la prueba del Modo Giroscopio. Fuente: Propia

Del mismo modo que ocurría en el caso anterior, al realizar una orientación con Roll negativo (inclinación hacia la izquierda), el motor derecho obtiene mayor relevancia respecto al izquierdo. Debido a esto, el vehículo es capaz de realizar un giro a izquierdas en función del ángulo de Roll adoptado y de manera lineal. Las señales PWM generadas en esta prueba son las siguiente (Figura 138).

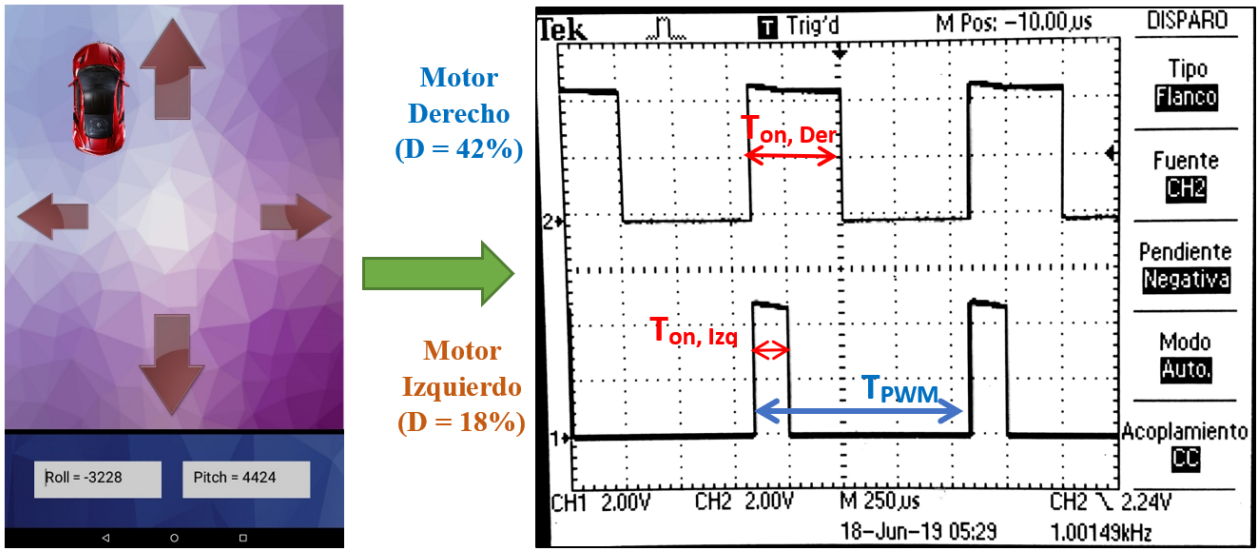


Figura 138: Inclinación en la aplicación móvil hacia Delante e Izquierda (Roll = -32,28º ; Pitch = 44,24º) y resultado de las PWM en los pines CCP1 y CCP2 del microcontrolador. Fuente: Propia

El ángulo Pitch adoptado otorgaría a ambos motores el mismo ciclo de trabajo, sin embargo, al estar presente un ángulo de Roll negativo de -32,28º la relevancia del motor izquierdo disminuye hasta un 43% frente al 100% del motor derecho.

$$DutyCicle_{Der} = DutyPitch \cdot \frac{100}{100} = 42\% \cdot 1 = \mathbf{42\%}$$

$$DutyCicle_{Izq} = DutyPitch \cdot \frac{43}{100} = 42\% \cdot 0{,}43 = \mathbf{18\%}$$

Finalmente, una vez terminadas todas las pruebas respecto a los modos “Libre” y “Giroscopio” se puede decir que el resultado ha sido el esperado. Por lo tanto, dado que todo a funcionado de manera correcta puede llevarse a cabo la parte final del proyecto, en la cuál se desarrolla la aplicación móvil para *smartphone* con ambas pantallas (Pantalla del “Modo Libre” y “Pantalla del Modo Giroscopio”).

11. Presupuesto del proyecto

Todo proyecto requiere de un coste para poder desarrollarlo. En este apartado se expondrán estos diferentes costes realizando el cálculo total de la inversión requerida para llevarlo a cabo.

Los costes de este proyecto pueden dividirse en dos categorías principales: Costes debidos a los Recursos Materiales utilizados y Costes de Recursos Humanos donde vienen presentes las horas de trabajo dedicadas por el autor del proyecto. Además, también se incluyen los Otros Costes Asociados, en los cuales se incluyen diferentes gastos del proyecto como es el coste de la tutora y la amortización del computador utilizado durante el desarrollo de este trabajo semestral.

11.1. Costes de Recursos Materiales

Los diferentes materiales utilizados para el desarrollo del proyecto se listan en la Tabla 5. Además, se indica el precio unitario, la cantidad utilizada y el coste total asociado a cada producto.

Material	Cantidad	Precio Unitario	Coste Total [€]
Chasis + Ruedas + Motores	1	23,28	23,28
Protoboard (Ariston)	1	4,10	4,10
PIC16F723A	1	1,70	1,70
L293D Motor Shield	1	3,50	3,50
PICkit3	1	42,91	42,91
Bluetooth HC-05	1	3,80	3,80
Pack de Cables	2	1,74	3,48
Pack 4 Pilas	1	3,60	3,60
Coste Total Recursos Materiales [€]			**86,37**

Tabla 5: Coste de los Recursos Materiales. Fuente: Propia

11.2. Coste de los Recursos Humanos

La cantidad de horas invertidas por el autor del proyecto han de ser remuneradas. Es por ello, que se recogen en la siguiente tabla (Tabla 6) las horas invertidas en cada una de las partes del proyecto así como el precio estipulado para cada una de ellas.

Material	horas	€/hora	Coste [€]
Estudio	30	15,00	450,00
Montaje	18	15,00	270,00
Programación	120	15,00	1800,00
Pruebas	60	15,00	900,00
Memoria	90	15,00	1350,00
Coste Total Recursos Humanos [€]			**4770**

Tabla 6: Costes de los Recursos Humanos (Costes de las horas invertidas por el autor del proyecto). Fuente: Propia

11.3. Otros Costes Asociados

Además de considerar los costes principales del proyecto, se han de tener en cuenta el gasto asociado a las horas de dedicación de la tutora, así como el precio del ordenador utilizado y su respectivo tiempo de amortización. Estos datos se recogen en la Tabla 7.

Tipo	Nº de horas dedicadas		Precio	Coste [€]
Dedicación tutora	10 horas		45 €/hora	450,00
Tipo	**Valor**	**Amortización**	**Uso**	**Coste [€]**
Ordenador	1.500 €	A 3 años	0,5 años	250,00
Coste Total Otros [€]				**700,00**

Tabla 7: Cálculo de Otros Costes Asociados al proyecto. Fuente: Propia

11.4. Costes Totales del Proyecto

Finalmente, en este apartado se realiza el cálculo del coste total que supone el desarrollo del proyecto (Tabla 8).

Coste Total Recursos Materiales	86,37 €
Coste Total Recursos Humanos	4.770,00 €
Coste Total Otros	700,00 €
COSTE TOTAL DEL PROYECTO [€]	**5.556,37 €**

Tabla 8: Costes Totales del Proyecto. Fuente: Propia

Conclusiones

El desarrollo de este trabajo de fin de grado ha resultado ser una experiencia compleja pero muy enriquecedora. Partiendo de la programación para realizar el control del vehículo frente a los estímulos generados por el usuario, para realizar diferentes tipos de movimientos, además de saber actuar frente a los objetos del entorno donde se encuentra.

Se puede afirmar que se han logrado todos los objetivos propuestos al inicio con un resultado muy satisfactorio. El diseño y montaje de las diferentes partes y módulos además del chasis del vehículo ha sido la parte más sencilla. Por el contrario, en la parte de programación de los diferentes módulos se encontraron bastantes dificultades, esto es debido a que es necesario la inicialización y buen funcionamiento de cada uno de los periféricos externos, así como la adaptación de del código fuente C para la interpretación de los datos enviados desde la aplicación móvil para *smartphone,* para finalmente realizar las acciones requeridas en los motores DC.

Otro punto a destacar ha sido, el buen resultado del control del coche frente a las órdenes enviadas por el usuario desde la aplicación móvil para *Android* mediante una serie de opciones. Además, el vehículo se consigue detener satisfactoriamente en el modo de conducción “Libre” frente a un objeto sin colisionar con él. Todo esto ha supuesto un gran aprendizaje en cuanto a conocimientos de electrónica e informática.

Es importante confirmar que hoy en día mediante las nuevas tecnologías, se están mejorando cada vez más los sistemas de control en la automoción eléctrica, por lo que haber realizado el proyecto en este tipo de ámbito ha permitido a mejorar sus conocimientos previos al autor de este proyecto.

Para acabar, aunque el resultado del trabajo ha sido satisfactorio, cabe mencionar que al vehículo se le podrían hacer un conjunto de mejoras de cara al futuro. Como por ejemplo, realizar una placa de circuito impreso (PCB) para mejorar las conexiones y poder optimizar el espacio utilizado. Por otro lado, se podría mejorar la velocidad de transmisión de datos entre la aplicación móvil y el vehículo para conseguir un control en tiempo real de los movimientos de este.

Agradecimientos

En primer lugar, querría agradecer la educación adquirida durante mis tres años y medio en el grado de Ingeniería Industrial por la Universidad Pública de Navarra (UPNA) y también el hecho de haberme brindado la oportunidad de acabar el grado en la Universidad Politécnica de Cataluña (UPC).

También quiero agradecer a la Escuela Técnica Superior de Ingeniería Industrial (ETSEIB) las facilidades y motivación otorgadas para desarrollar este proyecto final de grado y poder así ampliar mis conocimientos previos.

Finalmente, estoy gratamente satisfecho y contento de mi directora de proyecto Rosa Rodríguez Montañés por darme toda la ayuda que he necesitado y dedicarme parte de su tiempo.

Bibliografía

En este apartado se incluye la información de las diferentes fuentes que se han ido citando a lo largo de este documento.

Referencias bibliográficas

[1] Microchip Tecnology Inc. *PIC16(L)F722A/723A*. 14 de julio del 2015

[2] Microchip Tecnology Inc. *HI-TECH C for PIC10/12/16 User's Guide*. 26 de marzo del 2009

[3] Microchip Tecnology Inc. *PICkit3 In-Circuit Debugger/Programmer User's Guide*. 15 de abril del 2013

[4] ITead Studio. *HC-05. Bluetooth to Serial Port Module User's Guide*. 18 de junio del 2010

[5] Texas Instruments. *L293x Datasheet*. Septiembre del 1986, Revisado en enero del 2016

[6] Mouser Electronics. *Ultrasonic Ranging Module HC – SR04*.

Barcelona, a 20 de junio de 2019. **(FIRMA)**

Trabajo de Final de Grado

Grado en Ingeniería en Tecnologías Industriales

Diseño y desarrollo del sistema de control de un vehículo de tres ruedas

ANEXOS

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

ANEXOS

Anexo 1. CÓDIGO FUENTE C *Programa Principal*

Anexo 2. CÓDIGO FUENTE C *Archivo de Cabecera "uart.h"*

Anexo 3. PROGRAMACIÓN POR BLOQUES *Aplicación Móvil de AppInventor 2*

Anexo 4. DATASHEET *PIC16(L)F722A/723A*

Anexo 5. DATASHEET *PICkit 3*

Anexo 6. DATASHEET *Módulo Bluetooth HC-05*

Anexo 7. DATASHEET *L293x Motor Shield*

Anexo 8. DATASHEET *Módulo de Ultrasonidos HC-SR04*

Trabajo de Final de Grado

Grado en Ingeniería en Tecnologías Industriales

Diseño y desarrollo del sistema de control de un vehículo de tres ruedas

ANEXO 1:
CÓDIGO FUENTE C
Programa Principal

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

```
#include <htc.h>
#include "uart.h"

__CONFIG(BOREN_OFF & WDTE_OFF & CP_OFF & PLLEN_ON & FOSC_INTOSCIO);

// DEFINICION DE CONSTANTES UTILIZADAS POR EL COMPILADOR

#define _XTAL_FREQ 16000000

#define PITCH_H 6000
#define PITCH_L 500
#define ROLL_H 6000
#define ROLL_L 500

#define trigger RA0
#define echo RB5

// DEFINIR VARIABLES GLOBALES A UTILIZAR

unsigned char DADA, MODO = 'r', letra_M = 'z', duty_cicle_M;
int valor;
float ROLL,PITCH;

//-------------------------------------------------------------------------------------------------

//*************************************************************
//*************//FUNCIONES MODO GIROSCOPIO//****************
//*************************************************************

// Función de cálculo de valores ROLL y PITCH

void Calculo_Roll_Pitch(int dato){

unsigned char bits_SUP;         // Se mira la máscara del dato
                                // y en función de su valor se
        bits_SUP = dato>>14;    // obtiene si es ROLL o PITCH

        switch(bits_SUP){

            case 0:     //ROLL POSITIVO     // En los casos ROLL
            ROLL = dato;                    // el dato no se modifica
            break;

            case 255:   //ROLL NEGATIVO
            ROLL = dato;
            break;

            case 1:     //PITCH POSITIVO    // En los casos PITCH
            PITCH = dato - 16384;           // se resta/suma 16384
            break;

            case 254:   //PITCH NEGATIVO
            PITCH = dato + 16384;
            break;

        }

}
```

```
// Función de generación de señales PWM de los motores

void Generar_PWMs(float roll,float pitch){

float norm_D_100, norm_I_100, duty_PITCH;

//******* OFFSETS *****

roll = roll -179;
pitch = pitch -179;

//******* ROLL *********
        if(roll >= ROLL_L && roll <= ROLL_H){ // GIRO A DERECHAS REGULADO
        norm_D_100 = 100 - (roll*100/ROLL_H); // Disminución de velocidad en motor Izquierdo
        norm_I_100 = 100;
        }
        else if(roll > ROLL_H){ //GIRO A DERECHAS
        norm_D_100 = 0; // Motor Derecho parado
        norm_I_100 = 100;
        }
        else if(roll <= -ROLL_L && roll >= -ROLL_H){ // GIRO A IZQUIERDAS REGULADO
        norm_D_100 = 100;
        norm_I_100 = 100 - (roll*100/-ROLL_H); // Disminución de velocidad en motor Derecho
        }
        else if(roll < -ROLL_H){ //GIRO A IZQUIERDAS
        norm_D_100 = 100;
        norm_I_100 = 0; // Motor Izquierdo parado
        }
        else{   // SIN GIRO (AMBOS MOTORES AL 100%)
        norm_D_100 = 100;
        norm_I_100 = 100;
        }

//******* PITCH ********
        if(pitch >= PITCH_L && pitch <= PITCH_H){ // HACIA ADELANTE REGULADO
        RC3 = 0; RC0 = 0;   // PWM positiva
        duty_PITCH = (pitch*250/PITCH_H);
        }
        else if(pitch > PITCH_H){ //HACIA ADELANTE (100%)
        RC3 = 0; RC0 = 0;   //PWM positiva
        duty_PITCH = 250;
        }
        else if(pitch <= -PITCH_L && pitch >= -PITCH_H){ //HACIA ATRAS REGULADO
        RC3 = 1; RC0 = 1;   //PWM negativa
        duty_PITCH = (pitch*250/-PITCH_H);
        }
        else if(pitch < -PITCH_H){ //HACIA ATRAS (-100%)
        RC3 = 1; RC0 = 1;   //PWM negativa
        duty_PITCH = 250;
        }
        RC3 = 0; RC0 = 0;   //PWM positiva
        duty_PITCH = 0;
        }
//******* PWMs ********* // Se calculan los respectivos DutyCicles de cada motor
                         // teniendo en cuenta los valores de control calculados
                         // anteriormente por el ROLL y PITCH

    if((pitch >= 0) || (duty_PITCH == 0)){ // CASO HACIA ADELANTE
    CCPR1L = duty_PITCH*(norm_I_100/100); //Duty cicle Motor Izquierdo
    CCPR2L = duty_PITCH*(norm_D_100/100); //Duty cicle Motor Derecho
    }
    else if(pitch<0){   // CASO HACIA ATRAS
    CCPR1L = 250 - (duty_PITCH*norm_I_100/100); //Duty cicle Motor Izquierdo
    CCPR2L = 250 - (duty_PITCH*norm_D_100/100); //Duty cicle Motor Derecho
    }

}
//-------------------------------------------------------------------------------------------
```

```
//*************************************************************
//***************//FUNCIONES MODO LIBRE//*******************
//*************************************************************

void Modo_Manual(int dato){ // Calcula la trayectoria y velocidad del
                            // vehiculo en funcion del dato recibido
int DUTY_CCP;               // (de la slider-bar o de los botones)

    if(dato > 100)  letra_M = dato; // Datos de botones "letras"
    else    duty_cicle_M = dato;    // Datos de la slider-bar "0-100"

    DUTY_CCP = (duty_cicle_M*5)/2;  // Se adaptan los datos al rango de
                                    // valores del DutyCicle (0-250)

    switch(letra_M){ // TRAYECTORIAS EN FUNCION DE LAS LETRAS (t u v w, z "STOP")

        case 't':   //COCHE HACIA ADELANTE

            RC3 = 0; RC0 = 0;   //PWM positiva
            CCPR1L = DUTY_CCP;
            CCPR2L = DUTY_CCP; // A ambos motores se le aplica el DutyCicle
                               // de la slider-bar
        break;

        case 'u':   //COCHE HACIA ATRÁS

            RC3 = 1; RC0 = 1;   //PWM negativa
            CCPR1L = 250 - DUTY_CCP;
            CCPR2L = 250 - DUTY_CCP; // A ambos motores se le aplica el DutyCicle
                                     // de la slider-bar de forma inversa (-)
        break;

        case 'v':   //COCHE HACIA LA DERECHA

            if(RC0 == 0){ // CASO HACIA ADELANTE

                CCPR1L = DUTY_CCP;
                CCPR2L = 0; // Motor Derecho parado

            }else if (RC0 == 1){ // CASO HACIA ATRAS

                CCPR1L = 250 - DUTY_CCP;
                CCPR2L = 250; // Motor Derecho parado
            }
        break;

        case 'w':   //COCHE HACIA LA IZQUIERDA

            if(RC0 == 0){ // CASO HACIA ADELANTE

                CCPR1L = 0; // Motor Izquierdo parado
                CCPR2L = DUTY_CCP;

            }else if (RC0 == 1){ // CASO HACIA ATRAS

                CCPR1L = 250; // Motor Izquierdo parado
                CCPR2L = 250 - DUTY_CCP;
            }
        break;

        case 'z':   //COCHE PARADO

            RC3 = 0; RC0 = 0;   //PWM positiva
            CCPR1L = 0;
            CCPR2L = 0; //Ambos Motores PARADOS
        break;
    }
}
```

```
//*********************************************************
//***************//FUNCIONES PRINCIPALES//*****************
//*********************************************************

void setup(void){       //*******FUNCIÓN SETUP********

//OSCILADOR
OSCCON = 0x30; //Frecuencia a 16MHz

//MODULO UART
UART_Init(); // Se configura el módulo UART para trabajar a
             // a 9600 baudios

//INTERRUPCIONES
GIE=1; // Enable Interrupciones Globales
PEIE=1; // Enable Interrupciones Periféricos
RCIE=1; // Enable Interrupción Dato Recibido BT (RX)
TXIE=0; // Disable Interrupción Dato Enviado (TX)
RBIE=1; // Enable Interrupción PORT B cambio de flanco (ULTRASONIDOS)

ANSELA=0b00000000; //en el 16F723A la configuracion (entradas A 0 D) es con el registro ANSELA
ANSELB=0b00000000;

//MÓDULO BT
TRISCbits.TRISC7 = 1; // RX como INPUT
TRISCbits.TRISC6 = 0; // TX como OUTPUT

//PWM
APFCONbits.CCP2SEL = 0; // CCP2 en RC1
TRISC = 0b11110000;     // RC3:RC0 Outputs (MOTORES)

CCP1CON=0b00001100;     //MODO PWM y resolución del ciclo de trabajo de 8bits
CCP2CON=0b00001100;     //MODO PWM y resolución del ciclo de trabajo de 8bits
T2CON=0b00000011;       //Postdivisor nos da igual, Pre-scaler=16, TMR2ON=0
CCPR1L=0;               //Ton a 0
CCPR2L=0;               //Ton a 0

PR2 = 249;      //frecuencia de motores a 1000HZ
TMR2ON = 1; // Se activa el Timer 2

// ULTRASONIDOS
TRISAbits.TRISA0 = 0; // RA0 como OUTPUT (TRIGGER)
TRISBbits.TRISB5 = 1; // RB5 como INPUT (ECHO)
IOCBbits.IOCB5 = 1; // Se activa las Interrupciones por
                    // cambio de flanco del PORT B
T1CON=0b00110000;   // Pre-scaler=8, TMR1ON = 0

TMR1ON = 0; // Se mantiene parado el TIMER 1

}

void loop(void){        //********FUNCIÓN LOOP********

    while(1) {  // Bucle INFINITO

        if(MODO=='M'){      // En Modo Manual se envía
            trigger = 1;    // la señal de trigger del
            __delay_us(10); // sensor Ultrasonidos
            trigger = 0;
            __delay_ms(50); // Delay para dejar tiempo
        }                   // a que se reciba el "echo"
    }


}
```

```
static void interrupt RAI(void){    //********FUNCIÓN DE INTERRUPCIONES********

    if (PIR1bits.RCIF){ // INTERRUPCIÓN DATO RECIBIDO (RX)

                switch(MODO){ // Este Switch distigue el Modo en el que se encuentra
                              // el vehículo y lee los datos del buffer RCREG ("uart.h")

                    case 'G':   //***** GIROSCOPIO ******
                        valor = UART_Read_2byte(); // Lectura de 2 Bytes consecutivos
                        if(valor == 'r') MODO = 'r'; // Caso RESET (Botón BACK)
                    break;

                    case 'M':   //***** MANUAL ******
                        valor = UART_Read(); // Lectura de 1 Byte
                        if(valor == 'r') MODO = 'r'; // Caso RESET (Botón BACK)
                    break;
                                //***** RESET (Pantalla Menú Principal) ******
                    case 'r':   // 114 en ASCII
                        MODO = UART_Read(); // Lectura de 1 bytes (Modo)
                    break;
                }

                switch(MODO){ // Una vez leidos los datos en función del
                              // Modo, se aplica las funciones correspondientes

                    case 'G': // LLAMADA A FUNCIONES DEL MODO GIROSCOPIO
                        Calculo_Roll_Pitch(valor);
                        Generar_PWMs(ROLL,PITCH);
                    break;

                    case 'M': // LLAMADA A FUNCIONES DEL MODO LIBRE
                        Modo_Manual(valor);
                    break;

                    case 'r': // CASO RESET (MENU PRINCIPAL, TODO PARADO)
                    RC0=0; RC3=0; CCPR1L=0;CCPR2L=0;
                    break;
                }

    PIR1bits.RCIF = 0;  //Desactivamos la bandera de recepción en el buffer de entrada del USART

    }

    else if (INTCONbits.RBIF){ // INTERRUPCIÓN DEL ECHO DEL ULTRASONIDOS

        switch(echo){

            case 1: //Comienza el pulso de ECHO

                TMR1=0;
                TMR1ON=1; // El TIMER1 comienza a contar

            break;

            case 0: //Termina el pulso de ECHO

                TMR1ON=0; // El TIMER1 se detiene
                          // Temp_TMR1(us) = N*8*0,25us

                if(TMR1<300){ //Menos de 10,3cm (Dist_cm = Temp_TMR1/2*343*100)
                CCPR1L=0;CCPR2L=0; // Si la distancia es menor que 10 cm se para
                }

            break;

        }

        RBIF=0;  //Desactivamos la bandera de recepción de las int por cambio de flanco en el PORTB
    }

}
```

```
//-----------------------------------------------------------------------------------------------

//*********************************************************
//*****************\\Programa PRINCIPAL//*******************
//*********************************************************

void main(void){

    setup();                //Llamada a la función de configuración de registros

    loop();                 //Llamada a la función bucle, LOOP


}
```

Trabajo de Final de Grado

Grado en Ingeniería en Tecnologías Industriales

Diseño y desarrollo del sistema de control de un vehículo de tres ruedas

ANEXO 2:
CÓDIGO FUENTE C
Archivo de Cabecera "uart.h"

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

```
UART_Init(){

    TXSTA=0x20; // BRGH = 0
    SPBRG=25;   // Fosc = 16MHz, baud 9600 y BRGH = 0(MANUAL)
    RCSTA=0x90;
    //Baud Rate = Fosc/(64*(SPBRG+1))

  /*** Bits configurados de los registros TXSTA y RCSTA *******

    SYNC = 0;      // Selección del Modo Asíncrono del AUSART
    SPEN = 1;      // AUSART ON
    CREN = 1;      // Habilita la Recepción Continua
    TXEN = 1;      // Transmisión Habilitada
*/
}

void UART_Write(char data)
{
  while(!TRMT);
  TXREG = data;
}

char UART_TX_Empty()
{
  return TRMT;
}

void UART_Write_Text(char *text)
{
  int i;
  for(i=0;text[i]!='\0';i++)
    UART_Write(text[i]);
}

char UART_Data_Ready()
{
  return RCIF;
}

char UART_Read()
{
  while(!RCIF); // Mientras RCIF sea "0"
  return RCREG; // Retorna el byte de datos del buffer
}

void UART_Read_Text(char *Output, unsigned int length)
{
  unsigned int i;
  for(int i=0;i<length;i++)
  Output[i] = UART_Read();
}

int UART_Read_2byte()
{
  int Output = 0;

  while(!RCIF);       // Mientras RCIF sea "0"
  Output = RCREG;

  if (Output != 'r'){

    RCIF = 0;         // Se pone a 0 para esperar el
                      // siguiente Byte de datos
    while (!RCIF);
    Output+= RCREG<<8; // Se pone el siguiente Byte más significativo

  }

  return Output;
}
```

Trabajo de Final de Grado

Grado en Ingeniería en Tecnologías Industriales

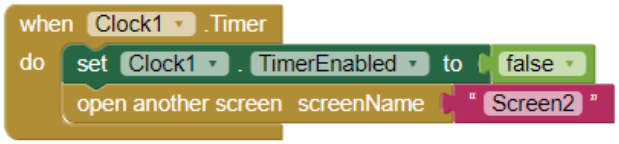
Diseño y desarrollo del sistema de control de un vehículo de tres ruedas

ANEXO 3: PROGRAMACIÓN POR BLOQUES

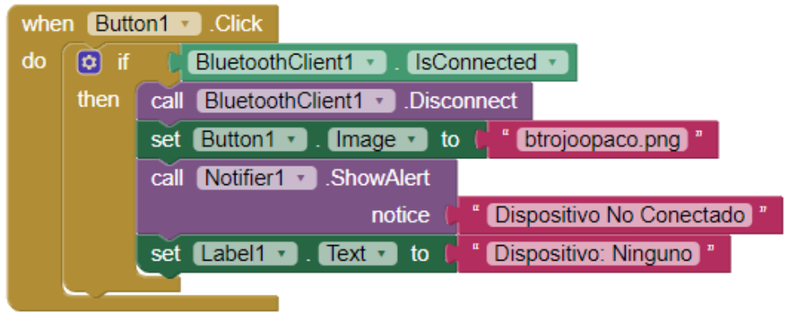
Aplicación Móvil de AppInventor 2

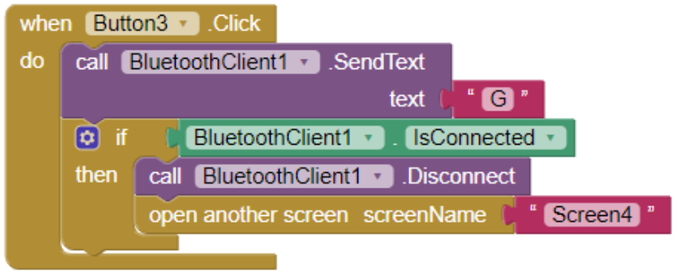
Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

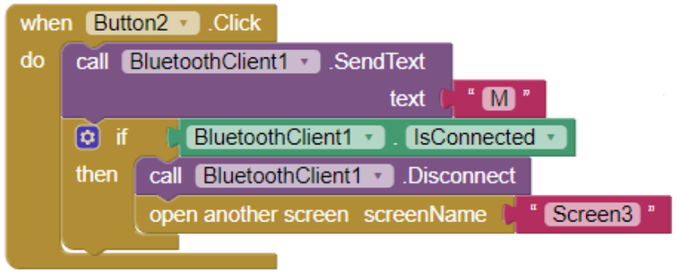
Pantalla de Inicio

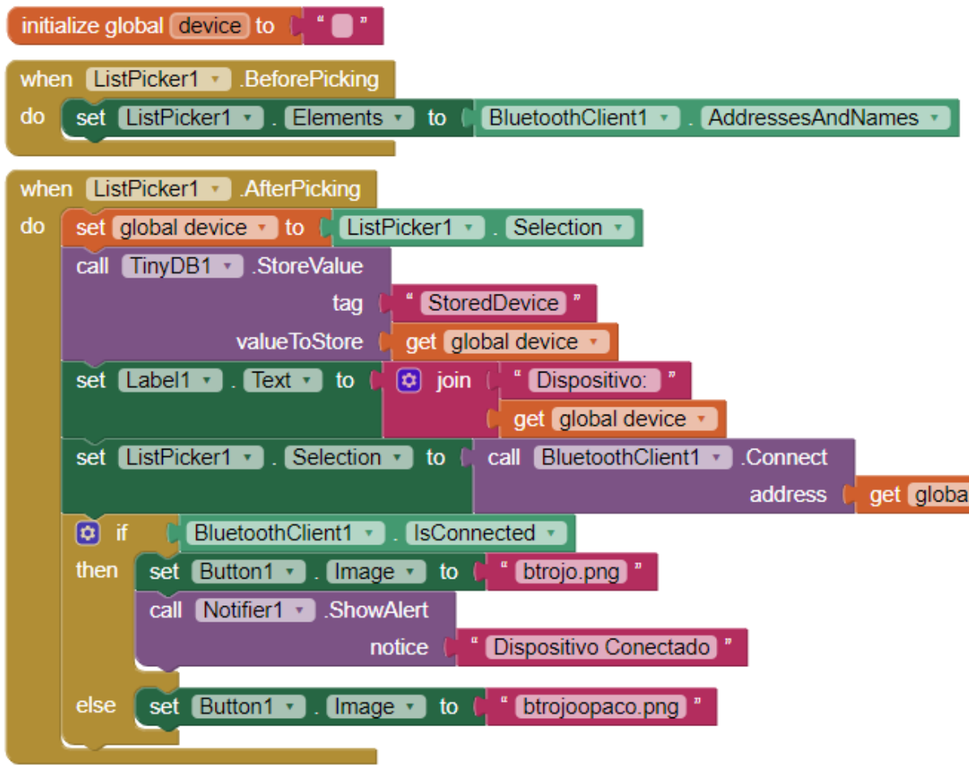


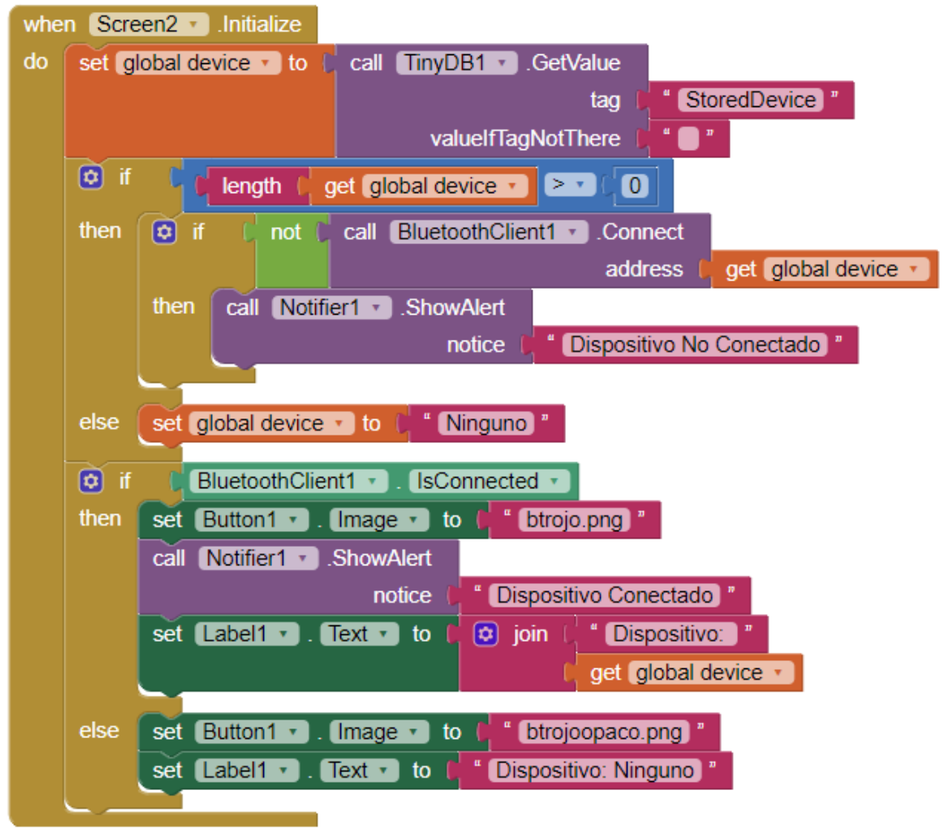
Pantalla de Menú Principal



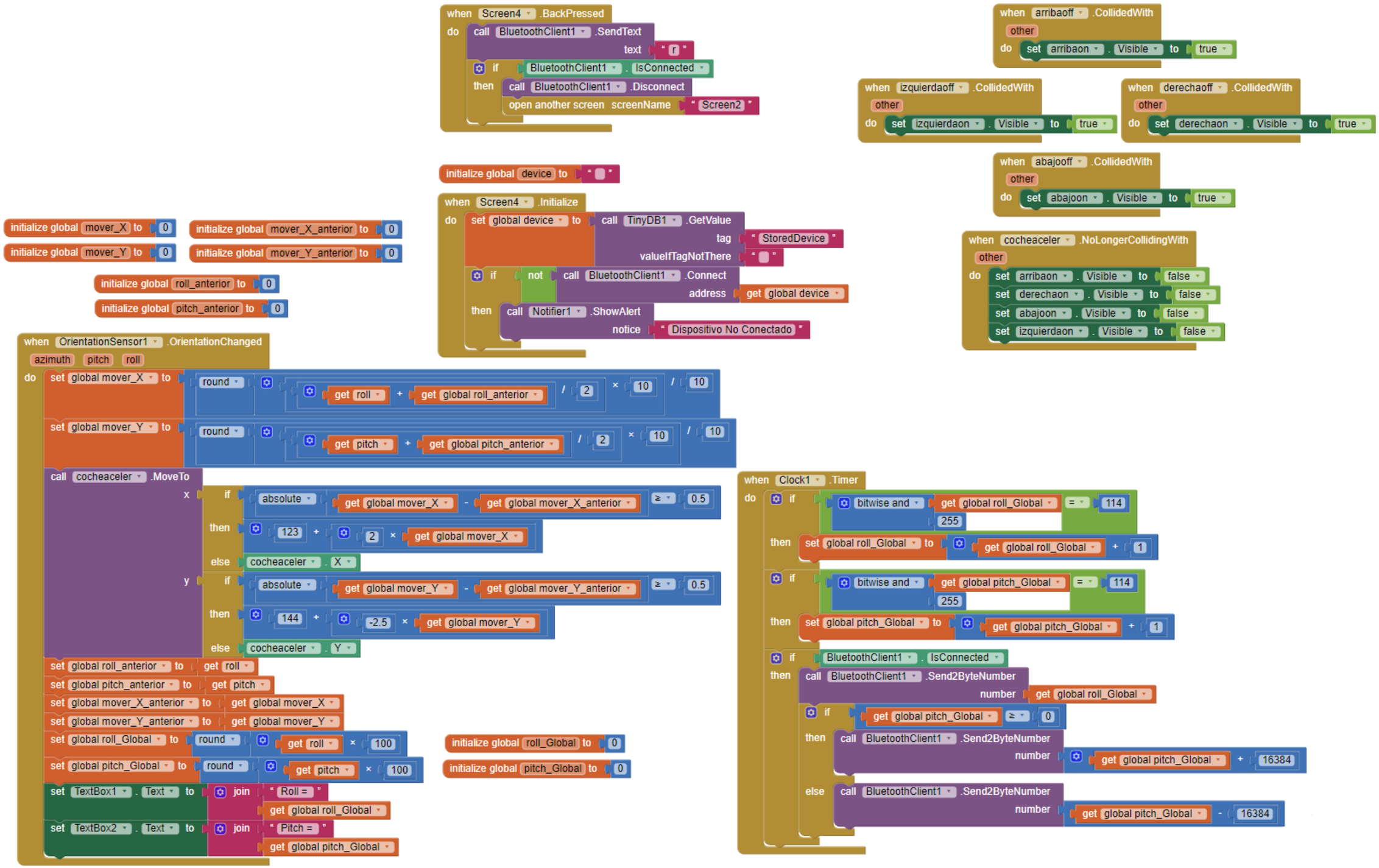




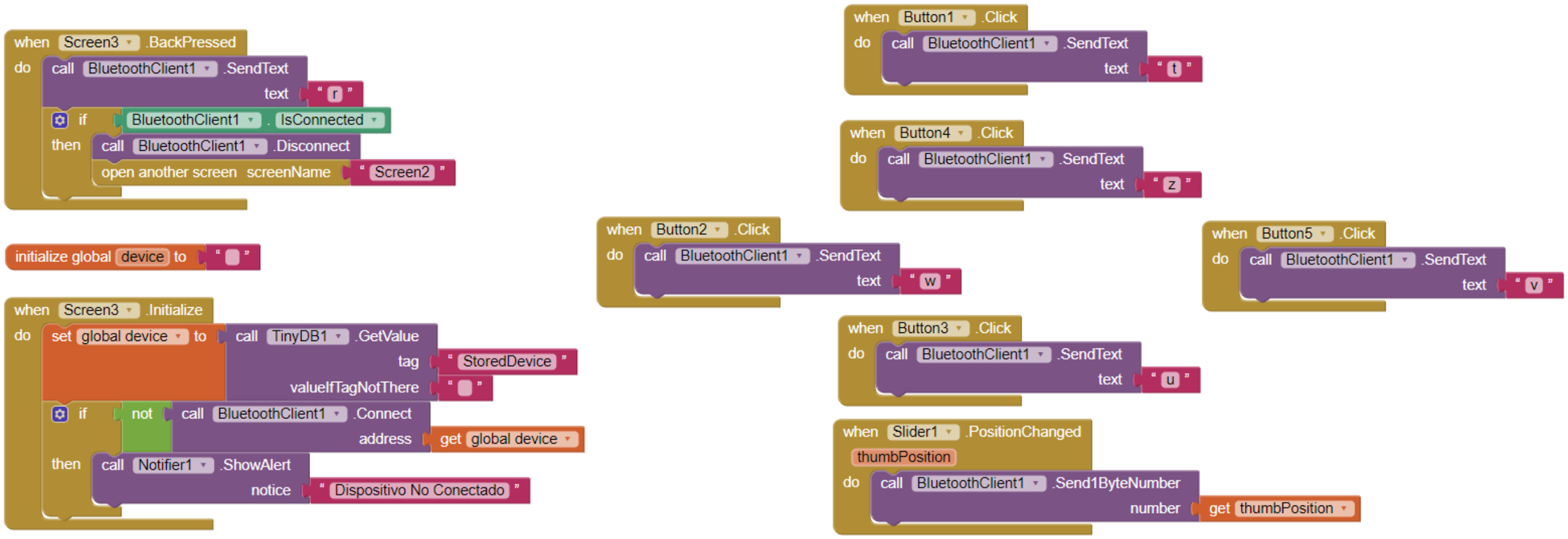




Pantalla del Modo Giroscopio



Pantalla del Modo Libre



Trabajo de Final de Grado

Grado en Ingeniería en Tecnologías Industriales

Diseño y desarrollo del sistema de control de un vehículo de tres ruedas

ANEXO 4:
DATASHEET
PIC16(L)F722A/723A

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

La DATASHEET *PIC16(L)F722A/723A* *es propiedad privada de la empresa "Microchip". El documento se adquiere de manera oficial a través de la siguiente página:*

http://ww1.microchip.com/downloads/en/DeviceDoc/40001417C.pdf

Trabajo de Final de Grado

Grado en Ingeniería en Tecnologías Industriales

Diseño y desarrollo del sistema de control de un vehículo de tres ruedas

ANEXO 5:
DATASHEET
PICkit3

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

La DATASHEET *PICkit 3* *es propiedad privada de la empresa "Microchip". El documento se adquiere de manera oficial a través de la siguiente página:*

https://ww1.microchip.com/downloads/en/DeviceDoc/51795B.pdf

Trabajo de Final de Grado

Grado en Ingeniería en Tecnologías Industriales

Diseño y desarrollo del sistema de control de un vehículo de tres ruedas

ANEXO 6:
DATASHEET
Módulo Bluetooth HC-05

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

HC-05

-Bluetooth to Serial Port Module

Overview

HC-05 module is an easy to use Bluetooth SPP (Serial Port Protocol) module, designed for transparent wireless serial connection setup.
Serial port Bluetooth module is fully qualified Bluetooth V2.0+EDR (Enhanced Data Rate) 3Mbps Modulation with complete 2.4GHz radio transceiver and baseband. It uses CSR Bluecore 04-External single chip Bluetooth system with CMOS technology and with AFH(Adaptive Frequency Hopping Feature). It has the footprint as small as 12.7mmx27mm. Hope it will simplify your overall design/development cycle.

Specifications

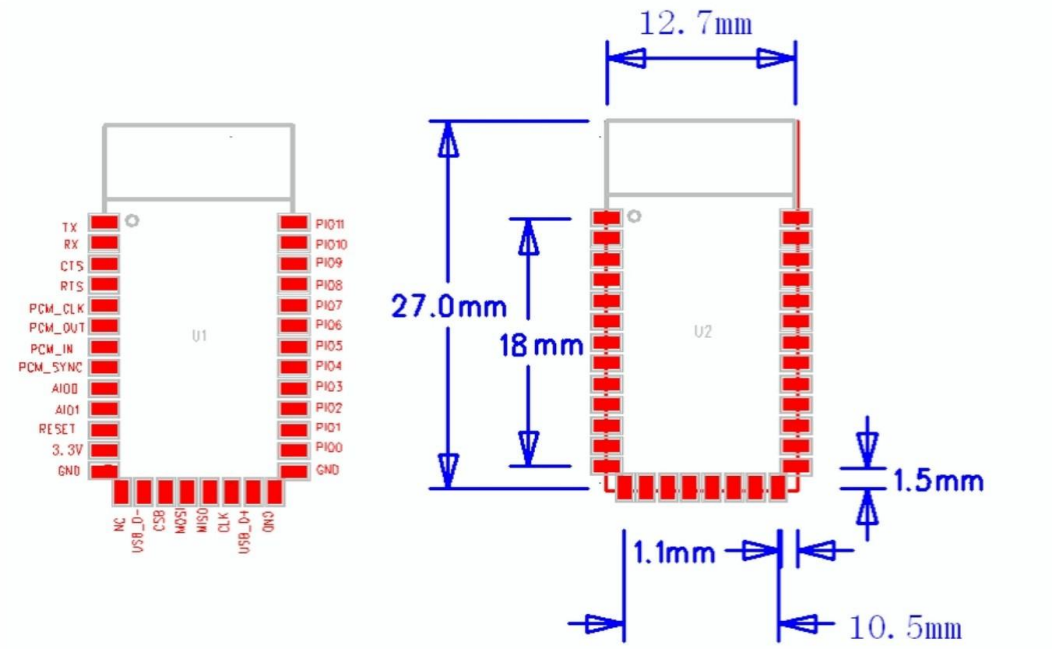
Hardware features

- Typical -80dBm sensitivity
- Up to +4dBm RF transmit power
- Low Power 1.8V Operation ,1.8 to 3.6V I/O
- PIO control
- UART interface with programmable baud rate
- With integrated antenna
- With edge connector

Software features

- Default Baud rate: 38400, Data bits:8, Stop bit:1,Parity:No parity, Data control: has. Supported baud rate: 9600,19200,38400,57600,115200,230400,460800.
- Given a rising pulse in PIO0, device will be disconnected.
- Status instruction port PIO1: low-disconnected, high-connected;
- PIO10 and PIO11 can be connected to red and blue led separately. When master and slave are paired, red and blue led blinks 1time/2s in interval, while disconnected only blue led blinks 2times/s.
- Auto-connect to the last device on power as default.
- Permit pairing device to connect as default.
- Auto-pairing PINCODE:"0000" as default
- Auto-reconnect in 30 min when disconnected as a result of beyond the range of connection.

Hardware



PIN Name	PIN #	Pad type	Description	Note
GND	13 21 22	VSS	Ground pot	
3.3 VCC	12	3.3V	Integrated 3.3V (+) supply with On-chip linear regulator output within 3.15-3.3V	
AIO0	9	Bi-Directional	Programmable input/output line	
AIO1	10	Bi-Directional	Programmable input/output line	
PIO0	23	Bi-Directional RX EN	Programmable input/output line, control output for LNA(if fitted)	
PIO1	24	Bi-Directional TX EN	Programmable input/output line, control output for PA(if fitted)	
PIO2	25	Bi-Directional	Programmable input/output line	
PIO3	26	Bi-Directional	Programmable input/output line	
PIO4	27	Bi-Directional	Programmable input/output line	
PIO5	28	Bi-Directional	Programmable input/output line	
PIO6	29	Bi-Directional	Programmable input/output line	
PIO7	30	Bi-Directional	Programmable input/output line	
PIO8	31	Bi-Directional	Programmable input/output line	
PIO9	32	Bi-Directional	Programmable input/output line	
PIO10	33	Bi-Directional	Programmable input/output line	
PIO11	34	Bi-Directional	Programmable input/output line	

RESETB	**11**	CMOS input with weak internal pull-up	Reset if low.input debouncde so must be low for >5MS to cause a reset	
UART_RTS	**4**	**CMOS output, tri-stable with weak internal pull-up**	**UART request to send, active low**	
UART_CTS	**3**	**CMOS input with weak internal pull-down**	**UART clear to send, active low**	
UART_RX	**2**	**CMOS input with weak internal pull-down**	**UART Data input**	
UART_TX	**1**	**CMOS output, Tri-stable with weak internal pull-up**	**UART Data output**	
SPI_MOSI	**17**	**CMOS input with weak internal pull-down**	**Serial peripheral interface data input**	
SPI_CSB	**16**	**CMOS input with weak internal pull-up**	**Chip select for serial peripheral interface, active low**	
SPI_CLK	**19**	**CMOS input with weak internal pull-down**	**Serial peripheral interface clock**	
SPI_MISO	**18**	**CMOS input with weak internal pull-down**	**Serial peripheral interface data Output**	
USB_-	**15**	**Bi-Directional**		

USB_+	20	Bi-Directional		
NC	14			
PCM_CLK	5	Bi-Directional	Synchronous PCM data clock	
PCM_OUT	6	CMOS output	Synchronous PCM data output	
PCM_IN	7	CMOS Input	Synchronous PCM data input	
PCM_SYNC	8	Bi-Directional	Synchronous PCM data strobe	

AT command Default:

How to set the mode to server (master):
1. Connect PIO11 to high level.
2. Power on, module into command state.
3. Using baud rate 38400, sent the "AT+ROLE=1\r\n" to module, with "OK\r\n" means setting successes.
4. Connect the PIO11 to low level, repower the module, the module work as server (master).

AT commands: (all end with \r\n)

1. Test command:

Command	Respond	Parameter
AT	OK	-

2. Reset

Command	Respond	Parameter
AT+RESET	OK	-

3. Get firmware version

Command	Respond	Parameter
AT+VERSION?	+VERSION:<Param> OK	Param : firmware version

Example:
AT+VERSION?\r\n
+VERSION:2.0-20100601
OK

4. Restore default

Command	Respond	Parameter
AT+ORGL	OK	-

Default state:

Slave mode, pin code :1234, device name: H-C-2010-06-01 ,Baud 38400bits/s.

5. Get module address

Command	Respond	Parameter
AT+ADDR?	+ADDR:<Param> OK	Param: address of Bluetooth module

Bluetooth address: NAP: UAP : LAP

Example:

AT+ADDR?\r\n

+ADDR:1234:56:abcdef

OK

6. Set/Check module name:

Command	Respond	Parameter
AT+NAME=<Param>	OK	Param: Bluetooth module name (Default :HC-05)
AT+NAME?	+NAME:<Param> OK (/FAIL)	

Example:

AT+NAME=HC-05\r\n set the module name to “HC-05”

OK

AT+NAME=ITeadStudio\r\n

OK

AT+NAME?\r\n

+NAME: ITeadStudio

OK

7. Get the Bluetooth device name:

Command	Respond	Parameter
AT+RNAME?<Param1>	1. +NAME:<Param2> OK 2. FAIL	Param1,Param 2 : the address of Bluetooth device

Example: (Device address 00:02:72:od:22:24，name：ITead)

AT+RNAME? 0002，72，od2224\r\n

+RNAME:ITead

OK

8. Set/Check module mode:

Command	Respond	Parameter
AT+ROLE=<Param>	OK	Param: 0- Slave
AT+ ROLE?	+ROLE:<Param>	

	OK	1-Master 2-Slave-Loop

9. Set/Check device class

Command	Respond	Parameter
AT+CLASS=<Param>	OK	Param: Device Class
AT+ CLASS?	1. +CLASS:<Param> OK 2. FAIL	

10. Set/Check GIAC (General Inquire Access Code)

Command	Respond	Parameter
AT+IAC=<Param>	1.OK 2. FAIL	Param: GIAC (Default : 9e8b33)
AT+IAC	+IAC:<Param> OK	

Example:

AT+IAC=9e8b3f\r\n

OK

AT+IAC?\r\n

+IAC: 9e8b3f

OK

11. Set/Check -- Query access patterns

Command	Respond	Parameter
AT+INQM=<Param>,<Param2>,<Param3>	1.OK 2. FAIL	Param: 0——inquiry_mode_standard 1——inquiry_mode_rssi Param2: Maximum number of Bluetooth devices to respond to Param3: Timeout (1-48 : 1.28s to 61.44s)
AT+ INQM?	+INQM： <Param>,<Param2>,<Param3> OK	

Example:

AT+INQM=1,9,48\r\n

OK

AT+INQM\r\n

+INQM:1, 9, 48

OK

12. Set/Check PIN code:

Command	Respond	Parameter
AT+PSWD=<Param>	OK	Param: PIN code (Default 1234)
AT+ PSWD?	+ PSWD ： <Param> OK	

13. Set/Check serial parameter:

Command	Respond	Parameter
AT+UART=<Param>,<Param2>,<Param3>	OK	Param1: Baud Param2: Stop bit Param3: Parity
AT+ UART?	+UART=<Param>,<Param2>,<Param3> OK	

Example:

AT+UART=115200，1,2,\r\n

OK

AT+UART?

+UART:115200,1,2

OK

14. Set/Check connect mode:

Command	Respond	Parameter
AT+CMODE=<Param>	OK	Param: 0 - connect fixed address 1 - connect any address 2 - slave-Loop
AT+ CMODE?	+ CMODE:<Param> OK	

15. Set/Check fixed address:

Command	Respond	Parameter
AT+BIND=<Param>	OK	Param: Fixed address (Default 00:00:00:00:00:00)
AT+ BIND?	+ BIND:<Param> OK	

Example:

AT+BIND=1234，56，abcdef\r\n

OK

AT+BIND?\r\n

+BIND:1234:56:abcdef

OK

16. Set/Check LED I/O

Command	Respond	Parameter
AT+POLAR=<Param1,<Param2>	OK	Param1: 0- PIO8 low drive LED 1- PIO8 high drive LED
AT+ POLAR?	+ POLAR=<Param1>,<Param2> OK	

		Param2: 0- PIO9 low drive LED 1- PIO9 high drive LED

17. Set PIO output

Command	Respond	Parameter
AT+PIO=<Param1>,<Param2>	OK	Param1: PIO number Param2: PIO level 0- low 1- high

Example:

1. PIO10 output high level

AT+PI0=10， 1\r\n

OK

18. Set/Check – scan parameter

Command	Respond	Parameter
AT+IPSCAN=<Param1>,<Param2>,<Param3>,<Param4>	OK	Param1: Query time interval Param2：Query duration Param3：Paging interval Param4：Call duration
AT+IPSCAN?	+IPSCAN:<Param1>,<Param2>,<Param3>,<Param4> OK	

Example:

AT+IPSCAN =1234,500,1200,250\r\n

OK

AT+IPSCAN?

+IPSCAN:1234,500,1200,250

19. Set/Check – SHIFF parameter

Command	Respond	Parameter
AT+SNIFF=<Param1>,<Param2>,<Param3>,<Param4>	OK	Param1: Max time Param2: Min time Param3: Retry time Param4: Time out
AT+ SNIFF?	+SNIFF:<Param1>,<Param2>,<Param3>,<Param4> OK	

20. Set/Check security mode

Command	Respond	Parameter
AT+SENM=<Param1>,<Param2>	1. OK 2. FAIL	Param1: 0——sec_mode0+off 1——sec_mode1+non_se
AT+ SENM?	+ SENM:<Param1>,<Param2>	

	OK	cure 2——sec_mode2_service 3——sec_mode3_link 4——sec_mode_unknown Param2: 0——hci_enc_mode_off 1——hci_enc_mode_pt_to_pt 2——hci_enc_mode_pt_to_pt_and_bcast

21. Delete Authenticated Device

Command	Respond	Parameter
AT+PMSAD=<Param>	OK	Param: Authenticated Device Address

Example:

AT+PMSAD =1234,56,abcdef\r\n

OK

22. Delete All Authenticated Device

Command	Respond	Parameter
AT+ RMAAD	OK	-

23. Search Authenticated Device

Command	Respond	Parameter
AT+FSAD=<Param>	1. OK 2. FAIL	Param: Device address

24. Get Authenticated Device Count

Command	Respond	Parameter
AT+ADCN?	+ADCN：<Param> OK	Param: Device Count

25. Most Recently Used Authenticated Device

Command	Respond	Parameter
AT+MRAD?	+ MRAD：<Param> OK	Param: Recently Authenticated Device Address

26. Get the module working state

Command	Respond	Parameter

AT+ STATE?	+ STATE：<Param> OK	Param: “INITIALIZED” “READY” “PAIRABLE” “PAIRED” “INQUIRING” “CONNECTING” “CONNECTED” “DISCONNECTED” “NUKNOW”

27. Initialize the SPP profile lib

Command	Respond	Parameter
AT+INIT	1. OK 2. FAIL	-

28. Inquiry Bluetooth Device

Command	Respond	Parameter
AT+INQ	+INQ: <Param1>，<Param2>，<Param3> …. OK	Param1：Address Param2：Device Class Param3：RSSI Signal strength

Example:

AT+INIT\r\n

OK

AT+IAC=9e8b33\r\n

OK

AT+CLASS=0\r\n

AT+INQM=1,9,48\r\n

At+INQ\r\n

+INQ:2:72:D2224,3E0104,FFBC

+INQ:1234:56:0,1F1F,FFC1

+INQ:1234:56:0,1F1F,FFC0

+INQ:1234:56:0,1F1F,FFC1

+INQ:2:72:D2224,3F0104,FFAD

+INQ:1234:56:0,1F1F,FFBE

+INQ:1234:56:0,1F1F,FFC2

+INQ:1234:56:0,1F1F,FFBE

+INQ:2:72:D2224,3F0104,FFBC

OK

28. Cancel Inquiring Bluetooth Device

Command	Respond	Parameter
AT+ INQC	OK	-

29. Equipment Matching

Command	Respond	Parameter
AT+PAIR=<Param1>,<Param2>	1. OK 2. FAIL	Param1： Device Address Param2： Time out

30. Connect Device

Command	Respond	Parameter
AT+LINK=<Param>	1. OK 2. FAIL	Param： Device Address

Example:

AT+FSAD=1234,56,abcdef\r\n

OK

AT+LINK=1234,56,abcdef\r\n

OK

31. Disconnect

Command	Respond	Parameter
AT+DISC	1. +DISC:SUCCESS OK 2. +DISC:LINK_LOSS OK 3. +DISC:NO_SLC OK 4. +DISC:TIMEOUT OK 5. +DISC:ERROR OK	Param： Device Address

32. Energy-saving mode

Command	Respond	Parameter
AT+ENSNIFF=<Param>	OK	Param： Device Address

33. Exerts Energy-saving mode

Command	Respond	Parameter
AT+ EXSNIFF =<Param>	OK	Param： Device Address

Revision History

Rev.	Description	Release date
v1.0	Initial version	7/18/2010

Trabajo de Final de Grado

Grado en Ingeniería en Tecnologías Industriales

Diseño y desarrollo del sistema de control de un vehículo de tres ruedas

ANEXO 7:
DATASHEET
L293x Motor Shield

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

L293x Quadruple Half-H Drivers

1 Features

- Wide Supply-Voltage Range: 4.5 V to 36 V
- Separate Input-Logic Supply
- Internal ESD Protection
- High-Noise-Immunity Inputs
- Output Current 1 A Per Channel (600 mA for L293D)
- Peak Output Current 2 A Per Channel (1.2 A for L293D)
- Output Clamp Diodes for Inductive Transient Suppression (L293D)

2 Applications

- Stepper Motor Drivers
- DC Motor Drivers
- Latching Relay Drivers

3 Description

The L293 and L293D devices are quadruple high-current half-H drivers. The L293 is designed to provide bidirectional drive currents of up to 1 A at voltages from 4.5 V to 36 V. The L293D is designed to provide bidirectional drive currents of up to 600-mA at voltages from 4.5 V to 36 V. Both devices are designed to drive inductive loads such as relays, solenoids, DC and bipolar stepping motors, as well as other high-current/high-voltage loads in positive-supply applications.

Each output is a complete totem-pole drive circuit, with a Darlington transistor sink and a pseudo-Darlington source. Drivers are enabled in pairs, with drivers 1 and 2 enabled by 1,2EN and drivers 3 and 4 enabled by 3,4EN.

The L293 and L293D are characterized for operation from 0°C to 70°C.

Device Information(1)

PART NUMBER	PACKAGE	BODY SIZE (NOM)
L293NE	PDIP (16)	19.80 mm × 6.35 mm
L293DNE	PDIP (16)	19.80 mm × 6.35 mm

(1) For all available packages, see the orderable addendum at the end of the data sheet.

Logic Diagram

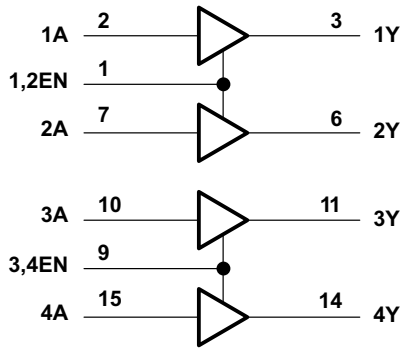


Table of Contents

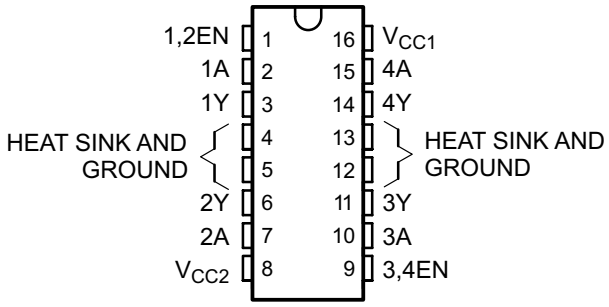
4 Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Revision C (November 2004) to Revision D	Page
• Removed *Ordering Information* table	1
• Added *ESD Ratings* and *Thermal Information* tables, *Feature Description* section, *Device Functional Modes*, *Application and Implementation* section, *Power Supply Recommendations* section, *Layout* section, *Device and Documentation Support* section, and *Mechanical, Packaging, and Orderable Information* section.	1

5 Pin Configuration and Functions

NE Package
16-Pin PDIP
Top View



Pin Functions

PIN		TYPE	DESCRIPTION
NAME	NO.		
1,2EN	1	I	Enable driver channels 1 and 2 (active high input)
<1:4>A	2, 7, 10, 15	I	Driver inputs, noninverting
<1:4>Y	3, 6, 11, 14	O	Driver outputs
3,4EN	9	I	Enable driver channels 3 and 4 (active high input)
GROUND	4, 5, 12, 13	—	Device ground and heat sink pin. Connect to printed-circuit-board ground plane with multiple solid vias
V_{CC1}	16	—	5-V supply for internal logic translation
V_{CC2}	8	—	Power VCC for drivers 4.5 V to 36 V

6 Specifications

6.1 Absolute Maximum Ratings

over operating free-air temperature range (unless otherwise noted)(1)

	MIN	MAX	UNIT
Supply voltage, V_{CC1}(2)		36	V
Output supply voltage, V_{CC2}		36	V
Input voltage, V_I		7	V
Output voltage, V_O	–3	V_{CC2} + 3	V
Peak output current, I_O (nonrepetitive, t ≤ 5 ms): L293	–2	2	A
Peak output current, I_O (nonrepetitive, t ≤ 100 µs): L293D	–1.2	1.2	A
Continuous output current, I_O: L293	–1	1	A
Continuous output current, I_O: L293D	–600	600	mA
Maximum junction temperature, T_J		150	°C
Storage temperature, T_{stg}	–65	150	°C

(1) Stresses beyond those listed under *Absolute Maximum Ratings* may cause permanent damage to the device. These are stress ratings only, which do not imply functional operation of the device at these or any other conditions beyond those indicated under *Recommended Operating Conditions*. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.
(2) All voltage values are with respect to the network ground terminal.

6.2 ESD Ratings

			VALUE	UNIT
$V_{(ESD)}$	Electrostatic discharge	Human-body model (HBM), per ANSI/ESDA/JEDEC JS-001(1)	±2000	V
		Charged-device model (CDM), per JEDEC specification JESD22-C101(2)	±1000	

(1) JEDEC document JEP155 states that 500-V HBM allows safe manufacturing with a standard ESD control process.
(2) JEDEC document JEP157 states that 250-V CDM allows safe manufacturing with a standard ESD control process.

6.3 Recommended Operating Conditions

over operating free-air temperature range (unless otherwise noted)

			MIN	NOM	MAX	UNIT
	Supply voltage	V_{CC1}	4.5		7	V
		V_{CC2}	V_{CC1}		36	
V_{IH}	High-level input voltage	V_{CC1} ≤ 7 V	2.3		V_{CC1}	V
		V_{CC1} ≥ 7 V	2.3		7	V
V_{IL}	Low-level output voltage		–0.3(1)		1.5	V
T_A	Operating free-air temperature		0		70	°C

(1) The algebraic convention, in which the least positive (most negative) designated minimum, is used in this data sheet for logic voltage levels.

6.4 Thermal Information

THERMAL METRIC(1)		L293, L293D NE (PDIP) 16 PINS	UNIT
$R_{\theta JA}$	Junction-to-ambient thermal resistance (2)	36.4	°C/W
$R_{\theta JC(top)}$	Junction-to-case (top) thermal resistance	22.5	°C/W
$R_{\theta JB}$	Junction-to-board thermal resistance	16.5	°C/W
ψ_{JT}	Junction-to-top characterization parameter	7.1	°C/W
ψ_{JB}	Junction-to-board characterization parameter	16.3	°C/W

(1) For more information about traditional and new thermal metrics, see the *Semiconductor and IC Package Thermal Metrics* application report, SPRA953.
(2) The package thermal impedance is calculated in accordance with JESD 51-7.

6.5 Electrical Characteristics

over operating free-air temperature range (unless otherwise noted)

PARAMETER			TEST CONDITIONS		MIN	TYP	MAX	UNIT
V_{OH}	High-level output voltage		L293: I_{OH} = −1 A		V_{CC2} – 1.8	V_{CC2} – 1.4		V
			L293D: I_{OH} = − 0.6 A					
V_{OL}	Low-level output voltage		L293: I_{OL} = 1 A			1.2	1.8	V
			L293D: I_{OL} = 0.6 A					
V_{OKH}	High-level output clamp voltage		L293D: I_{OK} = –0.6 A			V_{CC2} + 1.3		V
V_{OKL}	Low-level output clamp voltage		L293D: I_{OK} = 0.6 A			1.3		V
I_{IH}	High-level input current	A	V_I = 7 V			0.2	100	µA
		EN				0.2	10	
I_{IL}	Low-level input current	A	V_I = 0			–3	–10	µA
		EN				–2	–100	
I_{CC1}	Logic supply current		I_O = 0	All outputs at high level		13	22	mA
				All outputs at low level		35	60	
				All outputs at high impedance		8	24	
I_{CC2}	Output supply current		I_O = 0	All outputs at high level		14	24	mA
				All outputs at low level		2	6	
				All outputs at high impedance		2	4	

6.6 Switching Characteristics

over operating free-air temperature range (unless otherwise noted) V_{CC1} = 5 V, V_{CC2} = 24 V, T_A = 25°C

PARAMETER			TEST CONDITIONS	MIN	TYP	MAX	UNIT
t_{PLH}	Propagation delay time, low-to-high-level output from A input	L293NE, L293DNE	C_L = 30 pF, See Figure 2		800		ns
		L293DWP, L293N L293DN			750		
t_{PHL}	Propagation delay time, high-to-low-level output from A input	L293NE, L293DNE			400		ns
		L293DWP, L293N L293DN			200		
t_{TLH}	Transition time, low-to-high-level output	L293NE, L293DNE			300		ns
		L293DWP, L293N L293DN			100		
t_{THL}	Transition time, high-to-low-level output	L293NE, L293DNE			300		ns
		L293DWP, L293N L293DN			350		

6.7 Typical Characteristics

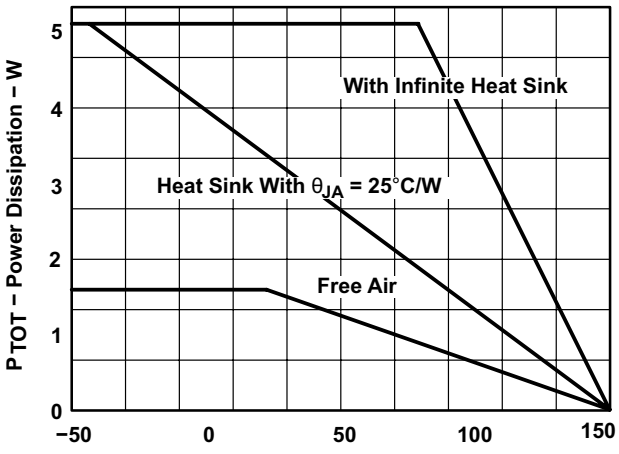
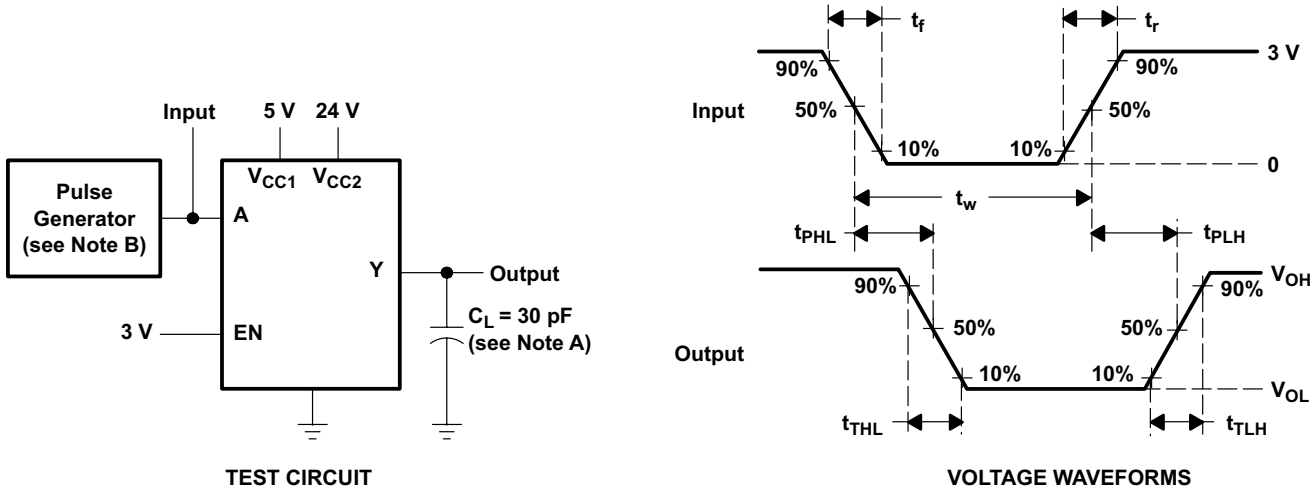


Figure 1. Maximum Power Dissipation vs Ambient Temperature

7 Parameter Measurement Information



TEST CIRCUIT **VOLTAGE WAVEFORMS**

NOTES: A. C_L includes probe and jig capacitance.
B. The pulse generator has the following characteristics: $t_r \leq 10$ ns, $t_f \leq 10$ ns, $t_w = 10$ µs, PRR = 5 kHz, $Z_O = 50$ Ω.

Figure 2. Test Circuit and Voltage Waveforms

8 Detailed Description

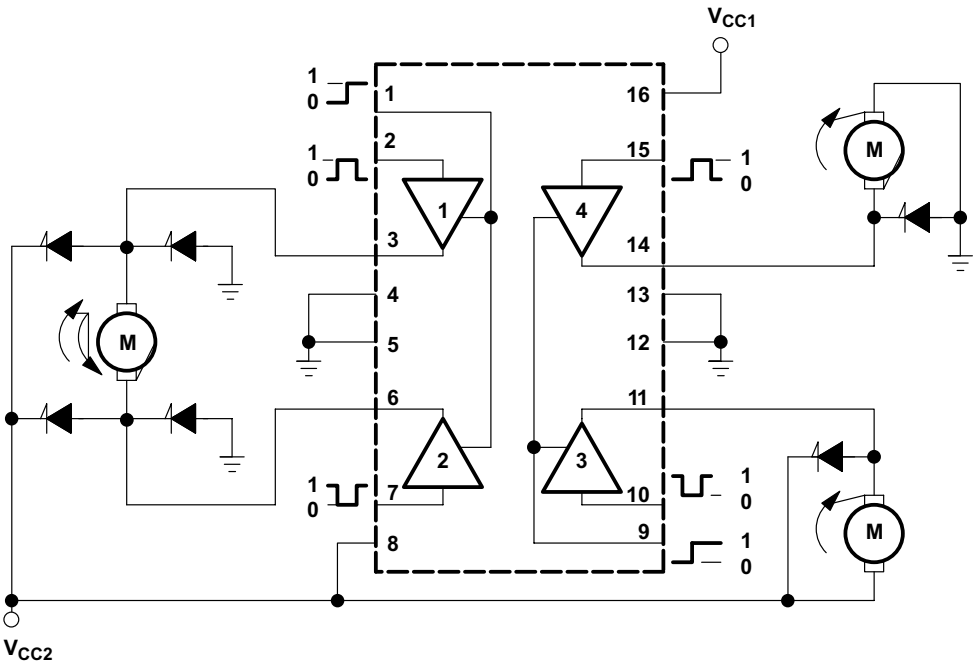
8.1 Overview

The L293 and L293D are quadruple high-current half-H drivers. These devices are designed to drive a wide array of inductive loads such as relays, solenoids, DC and bipolar stepping motors, as well as other high-current and high-voltage loads. All inputs are TTL compatible and tolerant up to 7 V.

Each output is a complete totem-pole drive circuit, with a Darlington transistor sink and a pseudo-Darlington source. Drivers are enabled in pairs, with drivers 1 and 2 enabled by 1,2EN and drivers 3 and 4 enabled by 3,4EN. When an enable input is high, the associated drivers are enabled, and their outputs are active and in phase with their inputs. When the enable input is low, those drivers are disabled, and their outputs are off and in the high-impedance state. With the proper data inputs, each pair of drivers forms a full-H (or bridge) reversible drive suitable for solenoid or motor applications.

On the L293, external high-speed output clamp diodes should be used for inductive transient suppression. On the L293D, these diodes are integrated to reduce system complexity and overall system size. A V_{CC1} terminal, separate from V_{CC2}, is provided for the logic inputs to minimize device power dissipation. The L293 and L293D are characterized for operation from 0°C to 70°C.

8.2 Functional Block Diagram



Output diodes are internal in L293D.

8.3 Feature Description

The L293x has TTL-compatible inputs and high voltage outputs for inductive load driving. Current outputs can get up to 2 A using the L293.

8.4 Device Functional Modes

Table 1 lists the fuctional modes of the L293x.

Table 1. Function Table (Each Driver)(1)

INPUTS (2)		OUTPUT (Y)
A	EN	
H	H	H
L	H	L
X	L	Z

(1) H = high level, L = low level, X = irrelevant, Z = high impedance (off)
(2) In the thermal shutdown mode, the output is in the high-impedance state, regardless of the input levels.

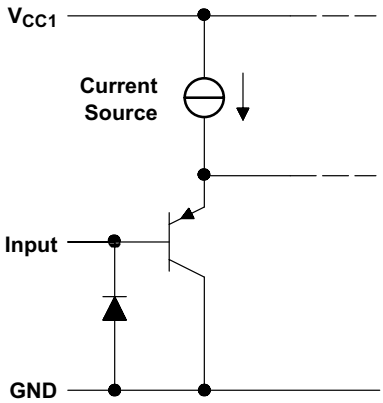


Figure 3. Schematic of Inputs for the L293x

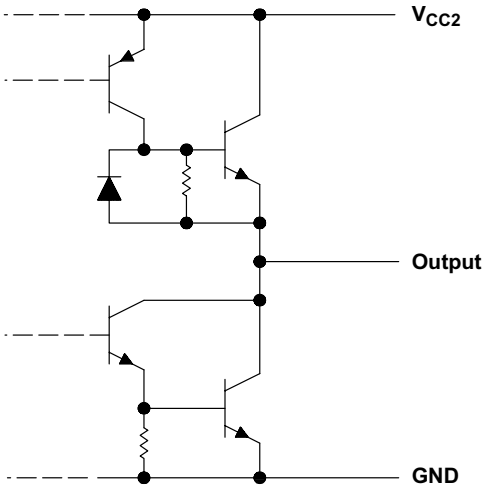


Figure 4. Schematic of Outputs for the L293

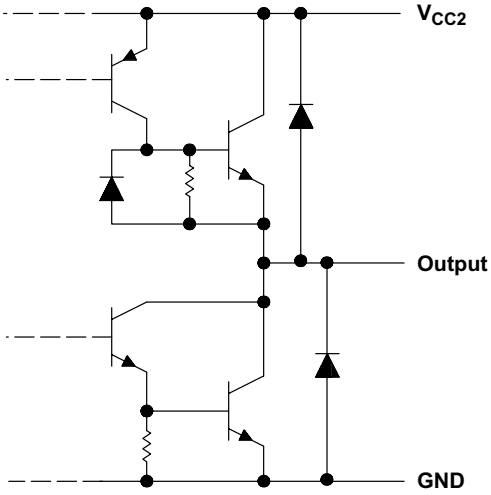


Figure 5. Schematic of Outputs for the L293D

9 Application and Implementation

NOTE

Information in the following applications sections is not part of the TI component specification, and TI does not warrant its accuracy or completeness. TI's customers are responsible for determining suitability of components for their purposes. Customers should validate and test their design implementation to confirm system functionality.

9.1 Application Information

A typical application for the L293 device is driving a two-phase motor. Below is an example schematic displaying how to properly connect a two-phase motor to the L293 device.

Provide a 5-V supply to V_{CC1} and valid logic input levels to data and enable inputs. V_{CC2} must be connected to a power supply capable of supplying the needed current and voltage demand for the loads connected to the outputs.

9.2 Typical Application

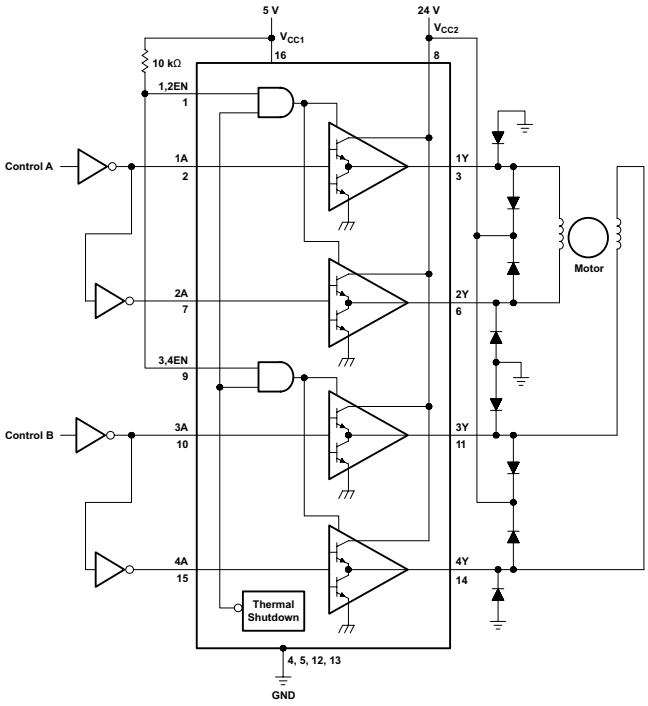


Figure 6. Two-Phase Motor Driver (L293)

9.2.1 Design Requirements

The design techniques in the application above as well as the applications below should fall within the following design requirements.

1. V_{CC1} should fall within the limits described in the *Recommended Operating Conditions*.
2. V_{CC2} should fall within the limits described in the *Recommended Operating Conditions*.
3. The current per channel should not exceed 1 A for the L293 (600mA for the L293D).

9.2.2 Detailed Design Procedure

When designing with the L293 or L293D, careful consideration should be made to ensure the device does not exceed the operating temperature of the device. Proper heatsinking will allow for operation over a larger range of current per channel. Refer to the *Power Supply Recommendations* as well as the *Layout Example*.

Typical Application (continued)

9.2.3 Application Curve

Refer to *Power Supply Recommendations* for additional information with regards to appropriate power dissipation. Figure 7 describes thermal dissipation based on Figure 14.

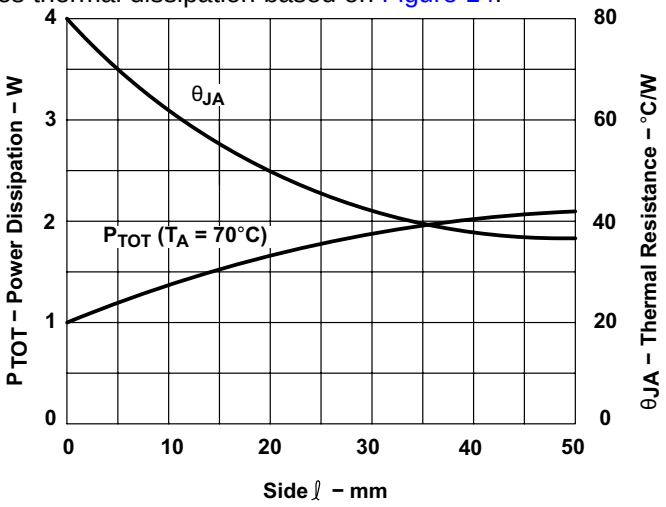


Figure 7. Maximum Power and Junction vs Thermal Resistance

9.3 System Examples

9.3.1 L293D as a Two-Phase Motor Driver

Figure 8 below depicts a typical setup for using the L293D as a two-phase motor driver. Refer to the *Recommended Operating Conditions* when considering the appropriate input high and input low voltage levels to enable each channel of the device.

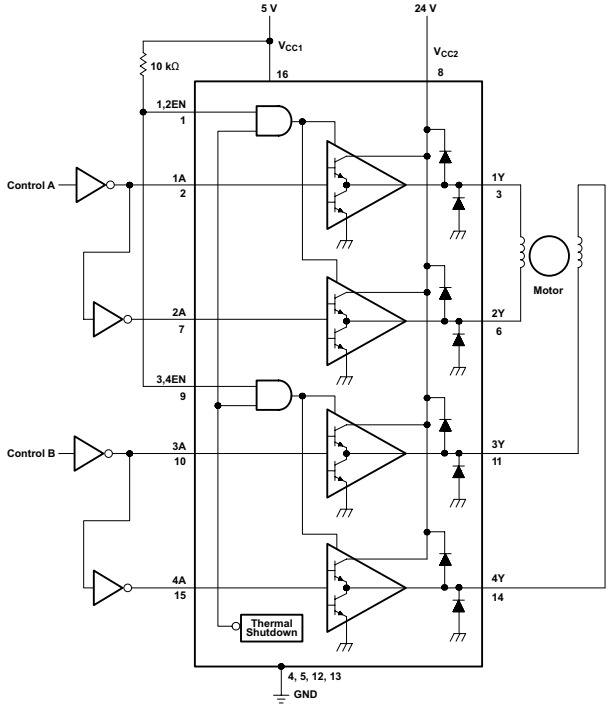
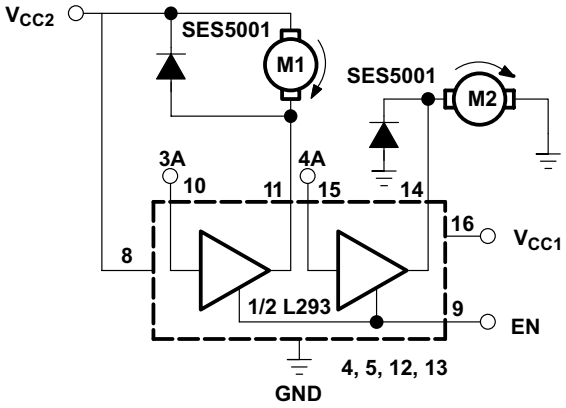


Figure 8. Two-Phase Motor Driver (L293D)

System Examples (continued)

9.3.2 DC Motor Controls

Figure 9 and Figure 10 below depict a typical setup for using the L293 device as a controller for DC motors. Note that the L293 device can be used as a simple driver for a motor to turn on and off in one direction, and can also be used to drive a motor in both directions. Refer to the function tables below to understand unidirectional vs bidirectional motor control. Refer to the *Recommended Operating Conditions* when considering the appropriate input high and input low voltage levels to enable each channel of the device.



Connections to ground and to supply voltage

Figure 9. DC Motor Controls

Table 2. Unidirectional DC Motor Control

EN	3A	M1 (1)	4A	M2
H	H	Fast motor stop	H	Run
H	L	run	L	Fast motor stop
L	X	Free-running motor stop	X	Free-running motor stop

(1) L = low, H = high, X = don't care

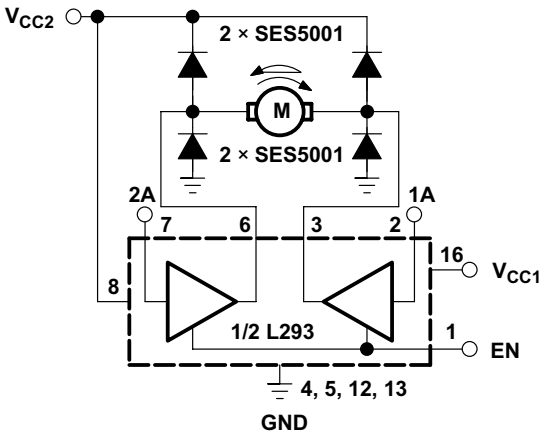


Figure 10. Bidirectional DC Motor Control

Table 3. Bidirectional DC Motor Control

EN	1A	2A	FUNCTION (1)
H	L	H	Turn right
H	H	L	Turn left

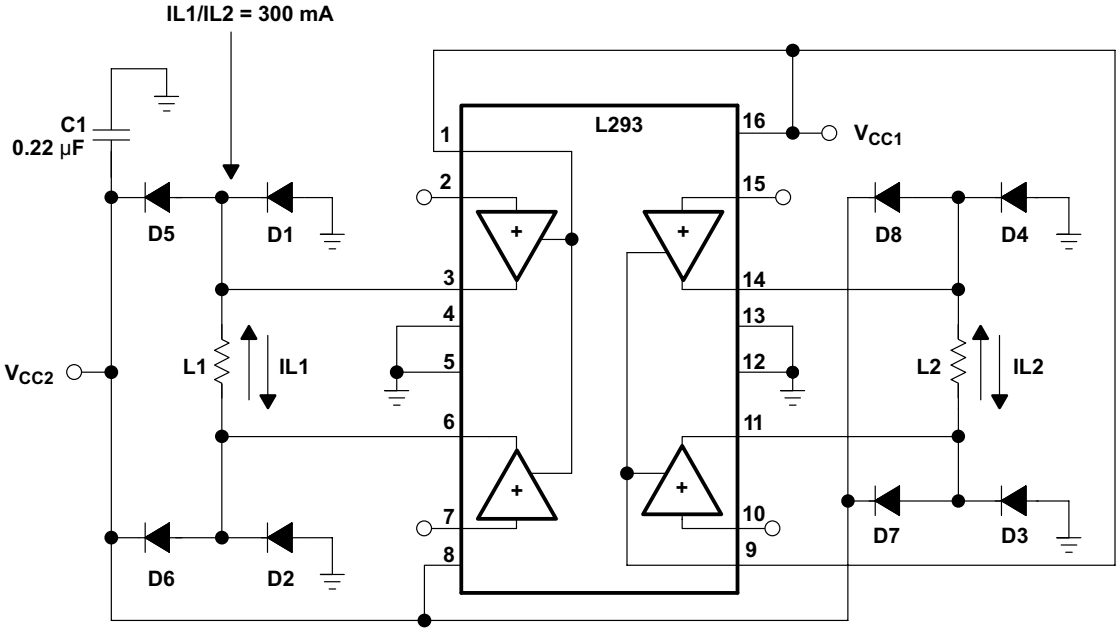
(1) L = low, H = high, X = don't care

Table 3. Bidirectional DC Motor Control (continued)

EN	1A	2A	FUNCTION(1)
H	L	L	Fast motor stop
H	H	H	Fast motor stop
L	X	X	Free-running motor stop

9.3.3 Bipolar Stepping-Motor Control

Figure 11 below depicts a typical setup for using the L293D as a two-phase motor driver. Refer to the *Recommended Operating Conditions* when considering the appropriate input high and input low voltage levels to enable each channel of the device.



D1−D8 = SES5001

Figure 11. Bipolar Stepping-Motor Control

10 Power Supply Recommendations

V_{CC1} is 5 V ± 0.5 V and V_{CC2} can be same supply as V_{CC1} or a higher voltage supply with peak voltage up to 36 V. Bypass capacitors of 0.1 uF or greater should be used at V_{CC1} and V_{CC2} pins. There are no power up or power down supply sequence order requirements.

Properly heatsinking the L293 when driving high-current is critical to design. The Rthj-amp of the L293 can be reduced by soldering the GND pins to a suitable copper area of the printed circuit board or to an external heat sink.

Figure 14 shows the maximum package power PTOT and the θJA as a function of the side of two equal square copper areas having a thickness of 35 µm (see Figure 14). In addition, an external heat sink can be used (see Figure 12).

During soldering, the pin temperature must not exceed 260°C, and the soldering time must not exceed 12 seconds.

The external heatsink or printed circuit copper area must be connected to electrical ground.

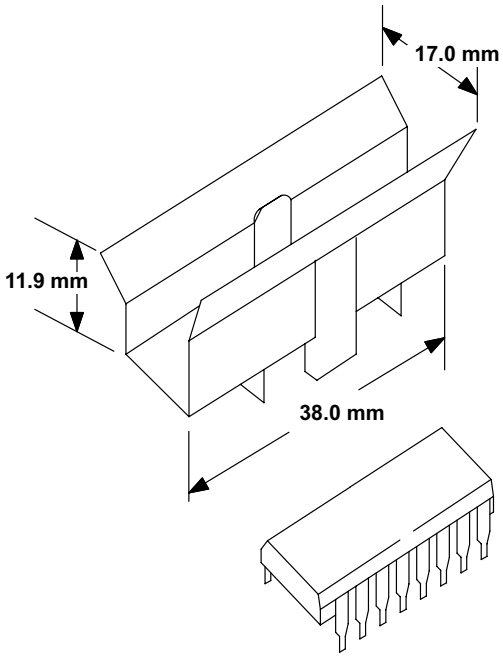


Figure 12. External Heat Sink Mounting Example (θ_{JA} = 25°C/W)

11 Layout

11.1 Layout Guidelines

Place the device near the load to keep output traces short to reduce EMI. Use solid vias to transfer heat from ground pins to ground plane of the printed-circuit-board.

11.2 Layout Example

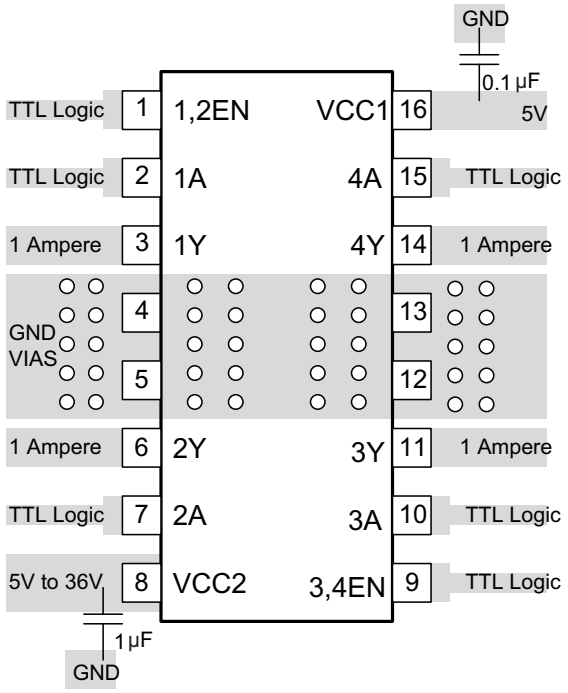


Figure 13. Layout Diagram

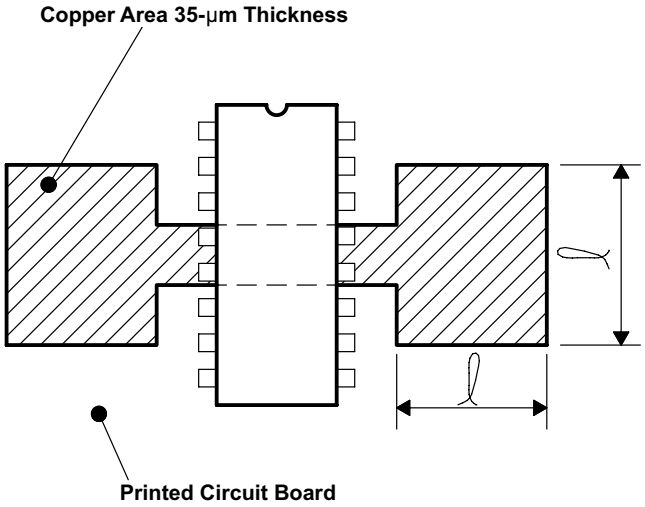


Figure 14. Example of Printed-Circuit-Board Copper Area (Used as Heat Sink)

12 Device and Documentation Support

12.1 Related Links

The table below lists quick access links. Categories include technical documents, support and community resources, tools and software, and quick access to sample or buy.

Table 4. Related Links

PARTS	PRODUCT FOLDER	SAMPLE & BUY	TECHNICAL DOCUMENTS	TOOLS & SOFTWARE	SUPPORT & COMMUNITY
L293	Click here	Click here	Click here	Click here	Click here
L293D	Click here	Click here	Click here	Click here	Click here

12.2 Community Resources

The following links connect to TI community resources. Linked contents are provided "AS IS" by the respective contributors. They do not constitute TI specifications and do not necessarily reflect TI's views; see TI's Terms of Use.

TI E2E™ Online Community ***TI's Engineer-to-Engineer (E2E) Community.*** Created to foster collaboration among engineers. At e2e.ti.com, you can ask questions, share knowledge, explore ideas and help solve problems with fellow engineers.

Design Support ***TI's Design Support*** Quickly find helpful E2E forums along with design support tools and contact information for technical support.

12.3 Trademarks

E2E is a trademark of Texas Instruments.
All other trademarks are the property of their respective owners.

12.4 Electrostatic Discharge Caution

These devices have limited built-in ESD protection. The leads should be shorted together or the device placed in conductive foam during storage or handling to prevent electrostatic damage to the MOS gates.

12.5 Glossary

SLYZ022 — *TI Glossary*.

This glossary lists and explains terms, acronyms, and definitions.

13 Mechanical, Packaging, and Orderable Information

The following pages include mechanical, packaging, and orderable information. This information is the most current data available for the designated devices. This data is subject to change without notice and revision of this document. For browser-based versions of this data sheet, refer to the left-hand navigation.

Trabajo de Final de Grado

Grado en Ingeniería en Tecnologías Industriales

Diseño y desarrollo del sistema de control de un vehículo de tres ruedas

ANEXO 8:
DATASHEET
Módulo de Ultrasonidos HC-SR04

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Ultrasonic Ranging Module HC - SR04

Product features:

Ultrasonic ranging module HC - SR04 provides 2cm - 400cm non-contact measurement function, the ranging accuracy can reach to 3mm. The modules includes ultrasonic transmitters, receiver and control circuit. The basic principle of work:
(1) Using IO trigger for at least 10us high level signal,
(2) The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back.
(3) IF the signal back, through high level , time of high output IO duration is the time from sending ultrasonic to returning.
Test distance = (high level time×velocity of sound (340M/S) / 2,

Wire connecting direct as following:

- 5V Supply
- Trigger Pulse Input
- Echo Pulse Output
- 0V Ground

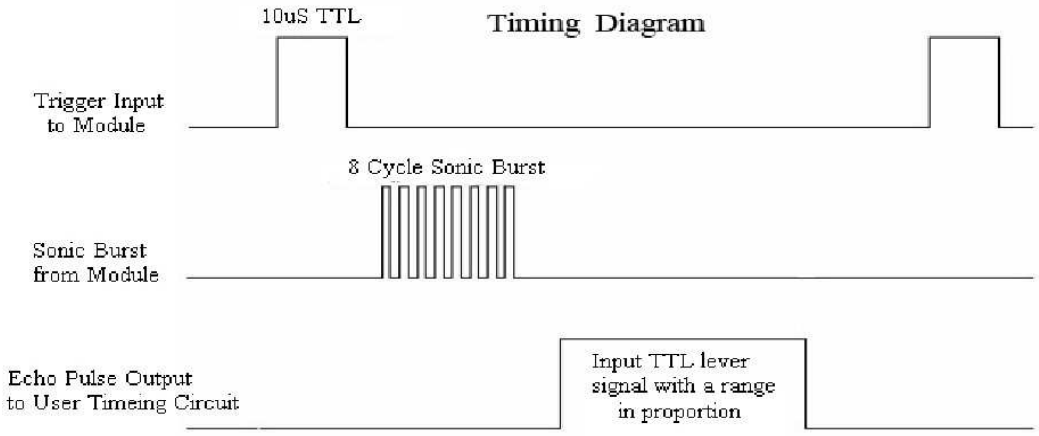
Electric Parameter

Working Voltage	**DC 5 V**
Working Current	**15mA**
Working Frequency	**40Hz**
Max Range	**4m**
Min Range	**2cm**
MeasuringAngle	**15 degree**
Trigger Input Signal	**10uS TTL pulse**
Echo Output Signal	**Input TTL lever signal and the range in proportion**
Dimension	**45*20*15mm**

Vcc Trig Echo GND

Timing diagram

The Timing diagram is shown below. You only need to supply a short 10uS pulse to the trigger input to start the ranging, and then the module will send out an 8 cycle burst of ultrasound at 40 kHz and raise its echo. The Echo is a distance object that is pulse width and the range in proportion .You can calculate the range through the time interval between sending trigger signal and receiving echo signal. Formula: uS / 58 = centimeters or uS / 148 =inch; or: the range = high level time * velocity (340M/S) / 2; we suggest to use over 60ms measurement cycle, in order to prevent trigger signal to the echo signal.



Attention:

- The module is not suggested to connect directly to electric, if connected electric, the GND terminal should be connected the module first, otherwise, it will affect the normal work of the module.
- When tested objects, the range of area is not less than 0.5 square meters and the plane requests as smooth as possible, otherwise ,it will affect the results of measuring.

www.Elecfreaks.com

Mouser Electronics

Authorized Distributor

Click to View Pricing, Inventory, Delivery & Lifecycle Information:

SparkFun Electronics:
SEN-13959